To the Graduate Council:

I am submitting herewith a thesis written by Mark Holdaway entitled "DC Servo Motor Robotic Drive System with Test Fixture." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Masters of Science, with a major in Electrical Engineering.

Robert E. Bodenheimer, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is make.

Requests for permission for extensive quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature Mark Haldaway

Date 12/2/94

# DC Servo Motor Robotic Drive System with Test Fixture

A Thesis

Presented for the

Master of Science

The University of Tennessee, Knoxville

Mark Holdaway

December, 1994

Copyright © Mark Holdaway, 1994
All rights reserved

# DEDICATIONS

This thesis is dedicated to my parents Ray and Rita Ann Holdaway for all love and support they gave me throughout my education.

# ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Bodenheimer, for his encouragement and patience. I would also like to thank the other committee members, Dr. Green and Dr Speckhart, for there comments and help. During the past three years, I have had the honor of working with several of the graduate students in the Mechanical and Aerospace department. I would never have completed my thesis without their encouragement and help. I would especially like to thank Mike Kennedy and Joel Lenoir for answering many questions about machine design. Also, I would like to thank Danny, Gary and Steve for all of their machining advice. I would like to thank Dennis for his help if finding parts for my thesis. I would like to thank Janine, Linda, and Betty for all of the help they have given me. Finally, I would like to thank my family for all of there support and encouragement.

# ABSTRACT

This thesis presents the design and construction of a test stand and a DC servo motor drive system. The drive system was designed to be compatible with both the hardware and the software of the Superior Engineering Design Program (SEDP) robot developed in the Mechanical and Aerospace Engineering Department of the University of Tennessee, Knoxville. A key design issues was to make the DC servo motor drive system directly interchangeable with the original stepping motor drive system in that it would accept the same drive commands as applied to the stepping motor driver. A switching amplifier was also designed that will power any DC motor with an average armature current of up to 12 amps operating as a closed-loop servo motor. The amplifier operates with any available supply voltage that is at least 8 volts higher than the nominal motor voltage.

# TABLE OF CONTENTS


CHAPTER PAGE

I. INTRODUCTION AND PURPOSE

Project Background . . . . . 1

Project Goals . . . . . 2

II. SYSTEM HARDWARE

System Overview . . . . . 5

PC Interface . . . . . 5

Quadrature Decoder . . . . . 10

|A-B| circuit . . . . . 15

Mechanical System . . . . . 16

User Interface . . . . . 18

PWM Circuit . . . . . 18

Drive Electronics . . . . . 22

Safety System . . . . . 26

Motor Model . . . . . 28

III. SYSTEM SOFTWARE

PC Interface . . . . . 30

|A-B| EPROM Programming . . . . . 35

User Interface . . . . . 36

PWM . . . . . 43

IV. SYSTEM INTEGRATION

Final Assembly . . . . . 47

Controls System Analysis . . . . . 48

V. CONCLUSIONS

Final Results . . . . . 57

Recommendations . . . . . 59

LIST OF REFERENCES ..... 62

APPENDIXES

Appendix A. Assembly Drawings ..... 65

Appendix B. System Modeling Calculations ..... 78

Appendix C. Program Listings ..... 91

VITA ..... 110

LIST OF TABLES


| TABLE | | PAGE |
|---|---|---|
| 2.1 | Interconnections between drive sub-systems | 7 |
| 2.2 | Truth table for quadrature decoder | 12 |
| 2.3 | Motor model variable parameters | 29 |
| 4.1 | Raw overshoot data for the original controller with no extra payload | 51 |
| 4.2 | Raw overshoot data for the original controller with a 5 lb payload | 52 |
| 4.3 | Raw overshoot data for the original controller with a 12 lb payload | 53 |
| 4.4 | Comparison of performance data for the system with and without the compensator | 56 |
| 5.1 | Cost summery of the drive system | 58 |

LIST OF FIGURES


| FIGURE | | PAGE |
|---|---|---|
| 1.1 | The "SEDP Robot" | 3 |
| 2.1 | Block diagram of sub-system interconnections in DC servo controller | 6 |
| 2.2 | PC interface between personal computer and drive system | 8 |
| 2.3 | Priority structure between multiple drives | 9 |
| 2.4 | Raw output of 400 count optical encoder | 11 |
| 2.5 | Sequential state machine operational diagram for quadrature decoder | 13 |
| 2.6 | Optical encoder, quadrature decoder, dual 16 bit counters, and \|A-B\| circuit | 14 |
| 2.7 | Machine drawing for mechanical slider | 17 |
| 2.8 | Motor mounting plate | 19 |
| 2.9 | User interface with LCD display to show status readout | 20 |
| 2.10 | Pulse width modulation feedback controller | 21 |
| 2.11 | H-Drive sub-system with IR2110 drive chips | 23 |
| 2.12 | H-Drive sub-system support electronics | 24 |
| 2.13 | Detailed close-up of the IR2110 level shifting chip | 25 |
| 2.14 | Safety system and power supplies | 27 |
| 2.15 | Block diagram of the motor | 29 |
| 3.1 | Flow diagram of the main section of the PC interface | 31 |
| 3.2 | Flow diagram of the step buffering subroutine | 32 |
| 3.3 | Flow diagram of the automatic homing subroutine | 34 |
| 3.4 | Flow diagram of the main section of the user interface | 37 |
| 3.5 | Flow diagram of the subroutine for position display | 38 |
| 3.6 | Flow diagram of the subroutine for displaying motor current | 40 |
| 3.7 | Flow diagram of the subroutine for setting desired motor current | 41 |
| 3.8 | Flow diagram of the subroutine for separating and printing one digit of a large number | 42 |

LIST OF FIGURES


| FIGURE | | PAGE |
|---|---|---|
| 3.9 | Flow diagram of the subroutine for displaying belt ratios | 44 |
| 3.10 | Flow diagram of the pulse width modulation controller program | 45 |
| 4.1 | Schematic of opto-isolator and op-amp buffer | 49 |
| 4.2 | Block diagram of the system | 54 |
| 4.3 | Step response of the system with and without the compensator | 55 |
| A.1 | Five faces of the corner blocks | 67 |
| A.2 | Machine drawing of the MOSFET's heatsink | 73 |
| A.3 | H-Drive circuit card and support circuitry | 75 |
| A.4 | Math card used to get \|A-B\| and dual 16 bit counters **A** and **B** | 76 |
| A.5 | User interface circuit card | 77 |
| A.6 | Circuit card layout for Quadrature decoder | 77 |

## LIST OF ABBREVIATIONS

| | |
|---|---|
| Pulse Width Modulation | PWM |
| Absolute value of A minus B | \|A-B\| |
| Superior Engineering Design Program | SEDP |
| Personal Computer | PC |
| Axis Zero Indicator | AZI |
| Clock | CLK |
| 16 bit counter for computer position | A |
| 16 bit counter for axis position | B |
| Electrically Programmable Read Only Memory | EPROM |
| Digital to Analog Converter | DAC |
| Liquid Crystal Digit | LCD |
| Bipolar Junction Transistor | BJT |
| Field Effect Transistor | FET |
| Metal Oxide Semiconductor | MOS |
| Ground Fault Interrupter | GFI |
| Axis Ready | AR |
| Transistor-Transistor Logic | TTL |
| American Standard Code for Information Interchange | ASCII |
| Electromagnetic Interference | EMI |
| Radio Frequency Interference | RFI |
| Advanced Continuous Simulation Language | ACSL |

# Chapter 1

# INTRODUCTION AND PURPOSE

## Project Background

Industrial robots, as a general rule, are expensive to purchase, setup, program and maintain. But yet, a customized robot offers accuracy and dependability not found in standard automation. Many small industries desire, and need, customized robots but cannot justify the cost burden; Petrocelli [1]. The absence, in the market place, of a low cost industrial robot was the motivation for the Superior Engineering Design Program (SEDP) group in selecting a goal of designing and building such a system. Relevant initial goals of the group are listed below.

- Provide an industrial robot to the customer, fully programmed for the initial application, for less than $15,000; McGuire [2]. This price should include installation, design of a custom end effector, and interfacing to the production line hardware.
- The industrial robot should be easily programmed so that robot motion paths and task operations can be modified by the customer.
- In order to minimize maintenance costs, the robot should be constructed of durable parts that can readily be located and purchased as off-the-self items.
- The robot should be reliable in order to avoid costly repairs and downtime that might otherwise make the system purchase economically unjustified.

The SEDP group designed and built an industrial robot that met the majority of these design goals. Figure 1.1 shows an isometric view of this system; Kennedy [3]. The original prototype unit, described by Williams [4], used a stepping motor drive in an open-loop control scheme. However, Moscone [5] points out that stepping motors generally have diminished output torque at one or more resonant frequencies. As a result, a stepping motor drive system often exhibits operation that is erratic, and perhaps dangerous, at speeds near these resonant points. If the motor is used at the resonant speeds, it can lose phase and the open-loop system will not be able to either detect or correct the position error. Nevertheless, many applications demand a high level of reliability and can not tolerate this large error. The potential for large positioning errors at operating speeds near the resonant frequencies of the stepping motor serves as a justification for designing and building a prototype system that eliminates this reliability problem.

## Project Goals

The need for greater reliability was the starting point for this thesis. The major goals for this work are:

- to expand the technology base for the SEDP group by providing a second motor drive system for the robot,
- to maintain software compatibility with the SEDP robot code and hardware,
- to create a portable teaching aid for the Controls and Smart Machinery classes, and

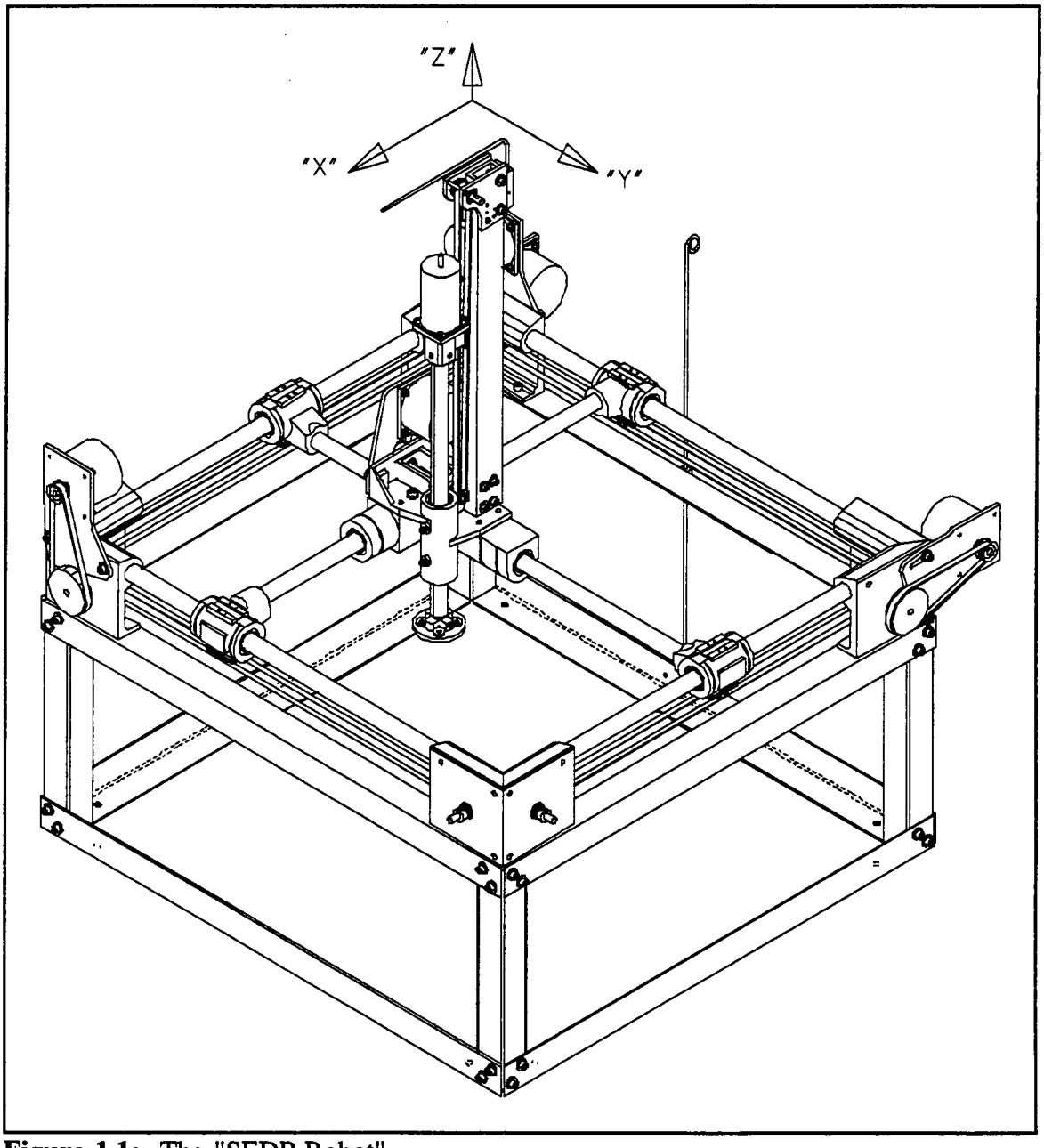



**Figure 1.1:** The "SEDP Robot".

Source: Michael Earl Kennedy, Low-Volume Production Design and Manufacture of a Low-Cost Industrial Robot, M.S. Thesis in Mechanical Engineering, The University of Tennessee, Knoxville, TN, May 1991, page 49.

- to create a drive system that will be a low cost, 'stand-alone' product that is usable with a variety of torque motors.

Development of the servo drive system is introduced in Chapter 2 where a description of the overall system and an explanation of the interconnections between sub-systems is given. This chapter also discusses each of the circuits in the drive sub-systems. Chapter 3 contains flow charts and descriptions of the software specifically developed for the drive. The final assembly is covered in Chapter 4, which also gives the results of the control system analysis. The final results and recommendations are covered in Chapter 5.

# Chapter 2

# SYSTEM HARDWARE

## System Overview

The DC servo drive system is made up of many interconnected sub-systems. Figure 2.1 shows a block diagram format of how each system is connected. Details of the links between sub-systems are provided in Table 2.1. Each of the sub-systems is explained in detail in the following sections. The first of these sections is the PC Interface, which provides communications with the outside world.

## PC Interface

The desired positional data has to have some means to get from the Personal Computer (PC) to the servo drive. Therefore a PC interface is needed. Both control information from the PC as well as information from other servo drives is converted by the PC interface into up and down count commands to the 16 bit counter named A. The PC interface also monitors the Axis Zero Indicator (AZI) to see if the slide is in the zero position. Inputs from the PC into the system are buffered though an optical isolator to protect the PC and user from a fault in the system. Table entries one through five in Table 2.1 describe each connection is detail. Figure 2.2 shows a diagram of the PC interface and Figure 2.3 shows an example of the priority structure for a four axes system.

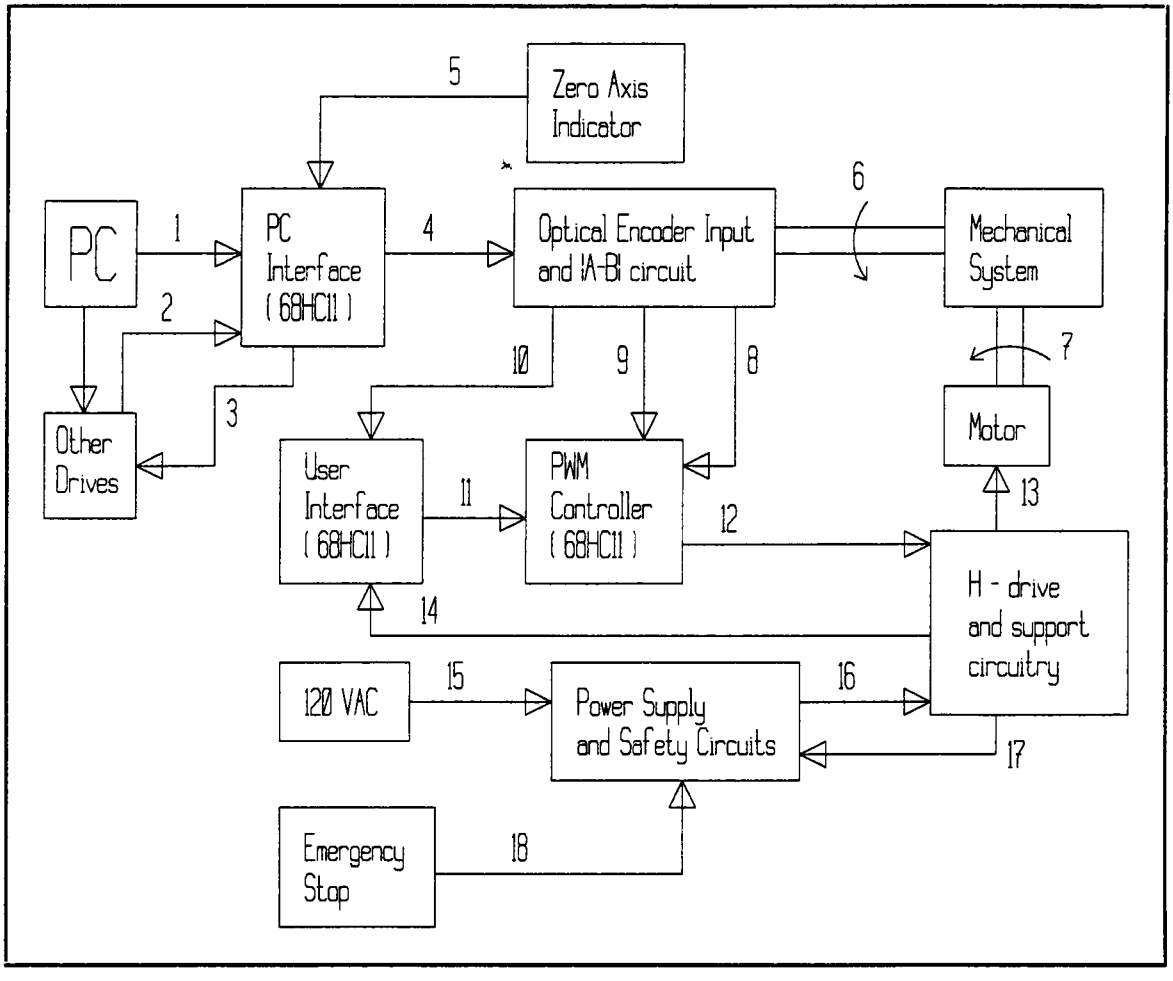



**Figure 2.1:** Block diagram of sub-system interconnections in the DC servo controller.

**Table 2.1:** Interconnections between drive sub-systems.

| # | Name | Type | Bits | Desc |
|---|---|---|---|---|
| 1 | Step<br>Direction<br>Reset<br>Axis Ready | Digital<br>Digital<br>Digital<br>Digital | 1<br>1<br>1<br>1 | Move by one step<br>Forward or Reverse<br>Move axis to 0 position and reset counters<br>Axis is home and ready to leave reset cond. |
| 2 | Reset | Digital | 1 | Move axis to 0 position and reset counters |
| 3 | Axis Ready | Digital | 1 | Axis is home and ready to leave reset cond. |
| 4 | UP<br>DOWN<br>RST_QUAD<br>RST_CNTR | Digital<br>Digital<br>Digital<br>Digital | 1<br>1<br>1<br>1 | Count up 1<br>Count down 1<br>Reset quadrature decoder<br>Reset dual 16 bit counters |
| 5 | AZI | Digital | 1 | Axis Zero Indicator - home position |
| 6 | shaft | Mech | n/a | Optical Encoder shaft |
| 7 | shaft | Mech | n/a | DC Motor shaft |
| 8 | A > B | Digital | 1 | controls direction of H-Drive |
| 9 | \| A-B \| | Digital | 12 | error signal - difference in positions |
| 10 | A | Digital | 16 | computer's desired position |
| 11 | TxD / RxD | Digital | 1 | Serial communication of drive current |
| 12 | PWM1<br>PWM2<br>$V_{ref}$ | Digital<br>Digital<br>Analog | 1<br>1<br>1 | control pulse to H-Drive for forward<br>control pulse to H-Drive for reverse<br>control voltage that $V_{Ia}$ is to track |
| 13 | motor lead | Digital | 1 | power lines to motor |
| 14 | $V_{Ia}$ | Analog | 1 | voltage proportional to armature current Ia |
| 15 | 120 VAC | Analog | 1 | input line voltage |
| 16 | $V_{HV}$ | Analog | 1 | 170 Volts to H-Drive |
| 17 | safety | Digital | 1 | over temperature or over current signal |
| 18 | safety | Digital | 1 | emergency stop |

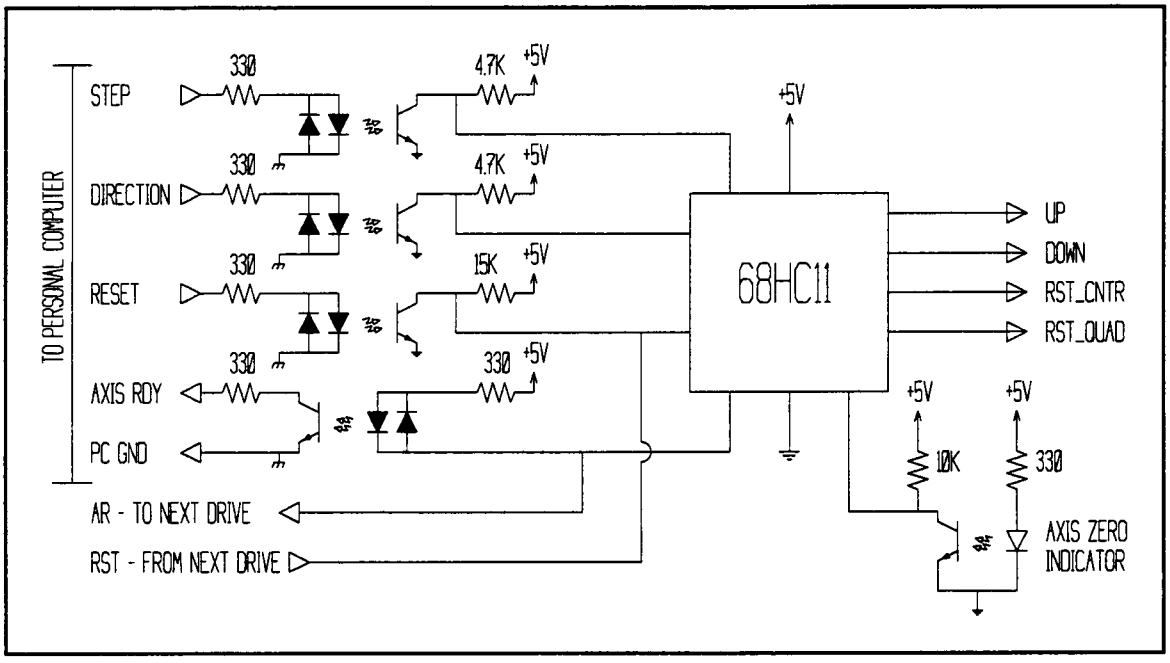



**Figure 2.2:** PC interface between personal computer and drive system.

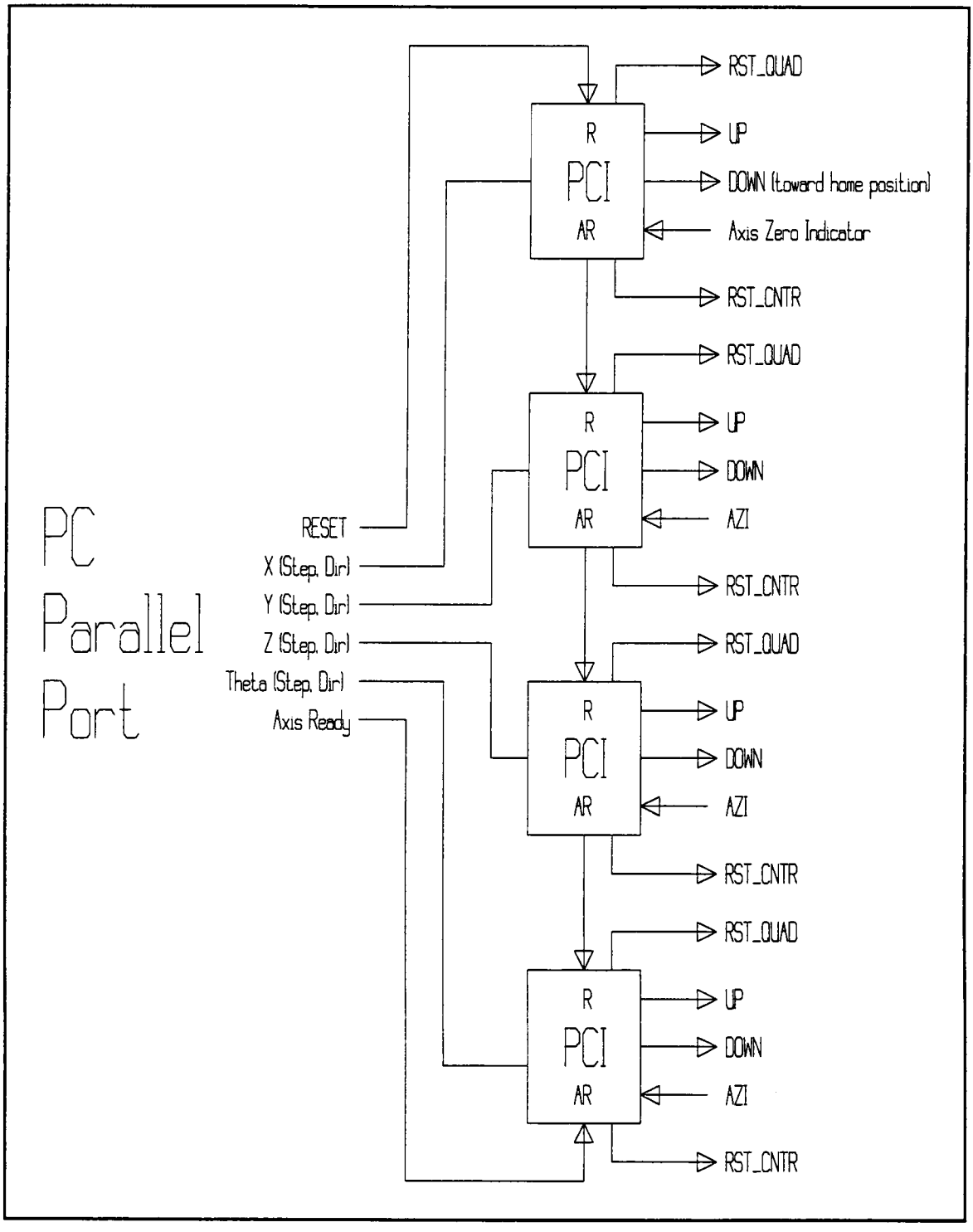



**Figure 2.3:** Priority structure between multiple drives.

Detailed information on the PC interface can be found in Chapter 3 which deals with the programs needed for each of the three 68HC11s used in the drive system.

## Quadrature Decoder

Positional data from the mechanical system is encoded by a 400 count optical encoder which generates two outputs, P1 and P2. Figure 2.4 depicts an example of P1 and P2 which are square waves that are 90° out of phase. However, the counting circuit expects a different type of input. The counting circuit expects one input to signal an up count and the other input to signal a down count. The repetitive patterns of P1 and P2 can be expressed in terms of a sequential state machine as seen in Figure 2.5. Table 2.2 shows how the information in the sequential state machine can be expressed as a truth table. From the truth table, the next state and output equations can be written and are show in Equations 2.1 through 2.4.

$$NextS_1 = P1 \cdot S_1 + P1 \cdot \overline{P2} \cdot \overline{S_0} + P1 \cdot P2 \cdot S_0 + P2 \cdot \overline{S_0} \cdot S1 + \overline{P2} \cdot S_0 \cdot S_1 \quad (2.1)$$

$$NextS_0 = P2 \cdot S_0 + P1 \cdot S_0 \cdot \overline{S_1} + P1 \cdot P2 \cdot S_1 + \overline{P1} \cdot S_0 \cdot S_1 + \overline{P1} \cdot P2 \cdot \overline{S_1} \quad (2.2)$$

$$CNTUP = P1 \cdot \overline{P2} \cdot \overline{S_0} \cdot \overline{S_1} \cdot CLK \quad (2.3)$$

$$CNTDOWN = \overline{P1} \cdot P2 \cdot \overline{S_0} \cdot \overline{S_1} \cdot CLK \quad (2.4)$$

The sequential state machine implementation can be seen in Figure 2.6 and includes the dual D flip-flops, the PAL, 1 MHz clock module, and the Schmitt trigger inverters. A single chip can be purchased for five dollars which can perform the same function.

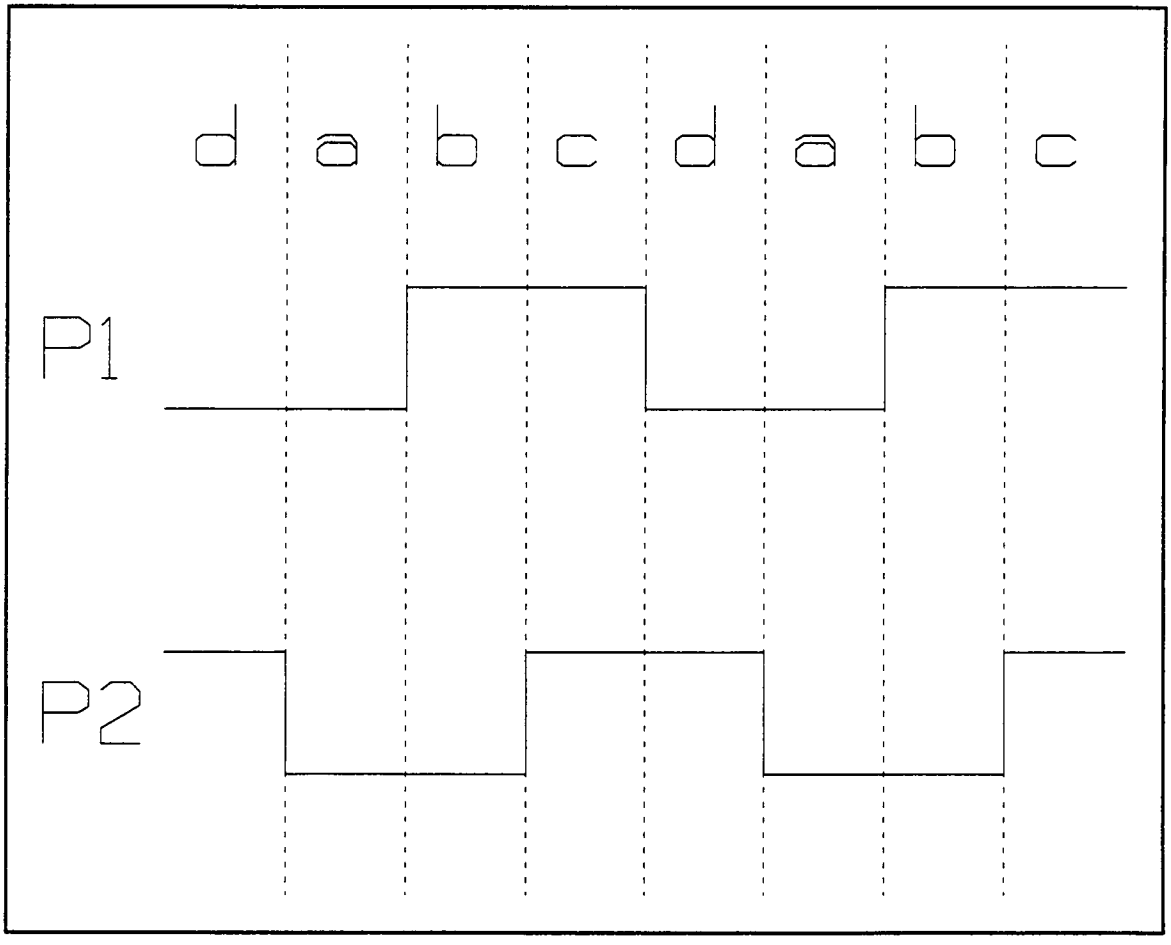



**Figure 2.4:** Raw output of 400 count optical encoder.

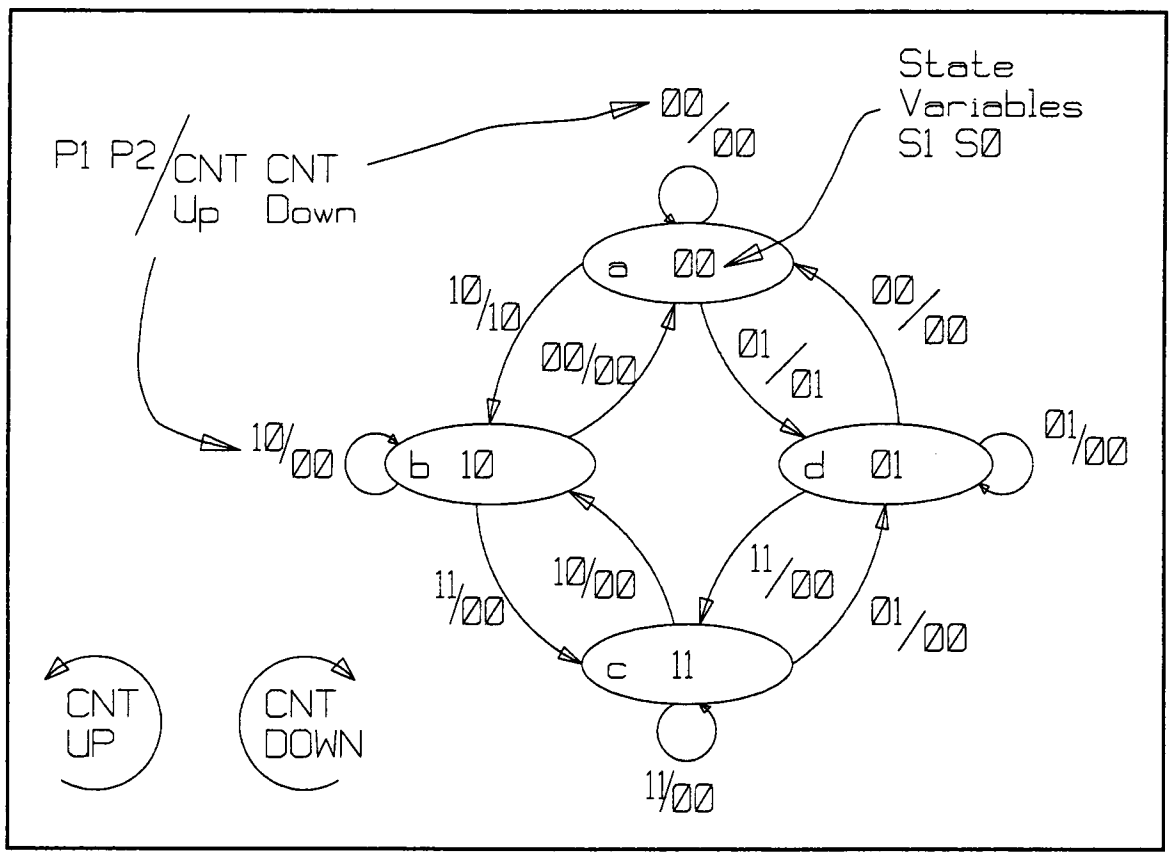



**Figure 2.5:** Sequential state machine operational diagram for quadrature decoder.

**Table 2.2:** Truth table for quadrature decoder

| | Present State | | | | Next State | | CNT | CNT |
|---|---|---|---|---|---|---|---|---|
| | S1 | S0 | P1 | P2 | S1 | S0 | UP | DOWN |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | | | | 1 | 0 | 1 | 0 | **1** |
| 2 | | | 1 | 0 | 1 | 0 | **1** | 0 |
| 3 | | | | 1 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | | | | 1 | 0 | 1 | 0 | 0 |
| 6 | | | 1 | 0 | 0 | 1 | 0 | 0 |
| 7 | | | | 1 | 1 | 1 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | | | | 1 | 1 | 0 | 0 | 0 |
| 10 | | | 1 | 0 | 1 | 0 | 0 | 0 |
| 11 | | | | 1 | 1 | 1 | 0 | 0 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 13 | | | | 1 | 0 | 1 | 0 | 0 |
| 14 | | | 1 | 0 | 1 | 0 | 0 | 0 |
| 15 | | | | 1 | 1 | 1 | 0 | 0 |

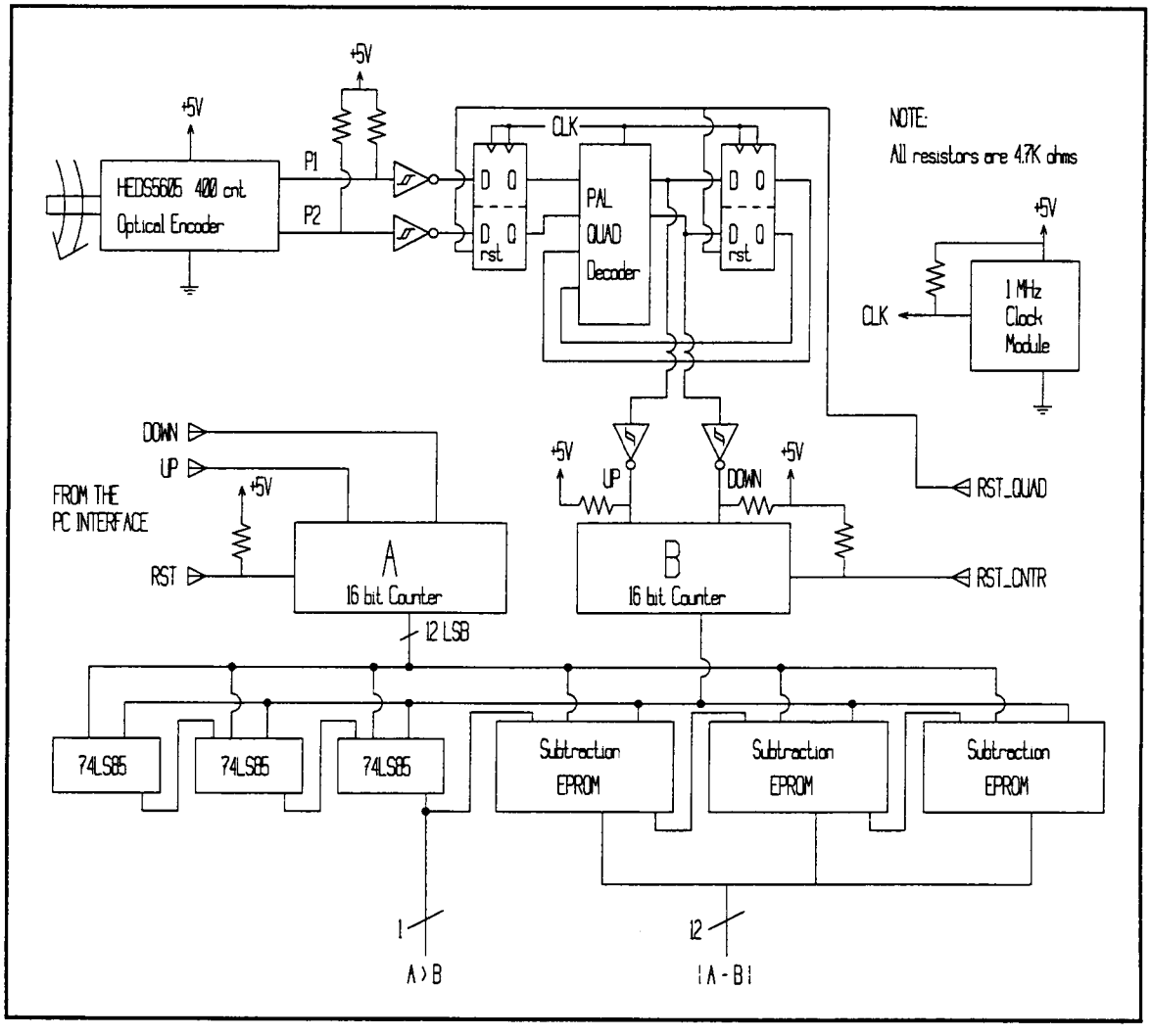



**Figure 2.6:** Optical encoder, quadrature decoder, dual 16 bit counters, and | A-B | circuit.

## | A-B | circuit

The up and down count commands from the optical encoder circuit are the input to the 16 bit counter named B. As mentioned earlier, the PC interface generates the up and down commands that go to the 16 bit counter named A. Figure 2.6 shows both the quadrature decoder, the counters(A and B) and math implementation. From A and B, the error signals **| A-B |** and **A>B** are formed. To satisfy the worse case, mechanically imposed speed requirement, the drive system must be able to respond to pulses at a rate of 8000 steps per second. Good control design recommends that the control system operate ten times faster than the system it controls. That makes the target chop speed 80 kHz or a time between pulses of 25 clock cycles. A clock cycle for the 68HC11 is one-half microsecond. Loading the two variables would take 37 clock cycles plus several more cycles to do the math. Therefore this math task could not be accomplished with the 68HC11. Using discrete logic chips the math could be completed within 0.15 microseconds. However, using discrete chips made the circuit prohibitively large. To make the circuit smaller, 3 EPROMS and 3 magnitude comparators were used to replace two banks of adders, three 2-1 multiplexer and several inverters. This version of the | A-B | circuit is smaller and has a calculation time of less than one microsecond. The EPROM version of the circuit was selected as the best solution based on it's speed and easier implementation. Even the selected version of the circuit is still very large and complicated; therefore, a large gate array would probably be a better choice for a production run of this sub-system.

## Mechanical System

A test stand was needed to provide a physical system on which the DC servo motor could operate. The stand would make it possible to test the drive system under varying load conditions as well as simulating a realistic operating environment. As a teaching aid, the entire system should be portable in order to be moved from class to class. Therefore, the stand was limited to a maximum weight of 50 pounds and a maximum length of 4 feet. In order to better simulate a realistic operating environment, the stand was patterned after one axis of the SEDP robot. With very few modifications, the test stand uses the same corner blocks, angle iron base, belting, pulleys, gears, and shafting as the SEDP robot. However, the axis slider used on the robot had to be redesigned to form a flat top on to which weight plates could be bolted onto. A new motor mounting that matched the profile of the DC motor had to be designed. Also, as the test stand was only a single axis, it was necessary to add an additional shaft to keep the slider block upright. Detailed machining instructions for the corner blocks and the slider can be found in Appendix A.

The redesigned slider, though totally different in appearance, shares many of the same critical distances as the original slider. A machine drawing of the slider is shown in Figure 2.7. The geometry of the one inch bearing hole and the belt channel is the most critical part of the slider. Also, extreme care must be taken in the final reaming of the bearing holes so that the bearings fit snugly. If the fit is too snug, the bearings may be damaged by excessive force during installation.

Redesigning the motor mounting plate was much easier. The mounting holes used to hold the plate to the corner block and outer profile were not changed. Only the motor

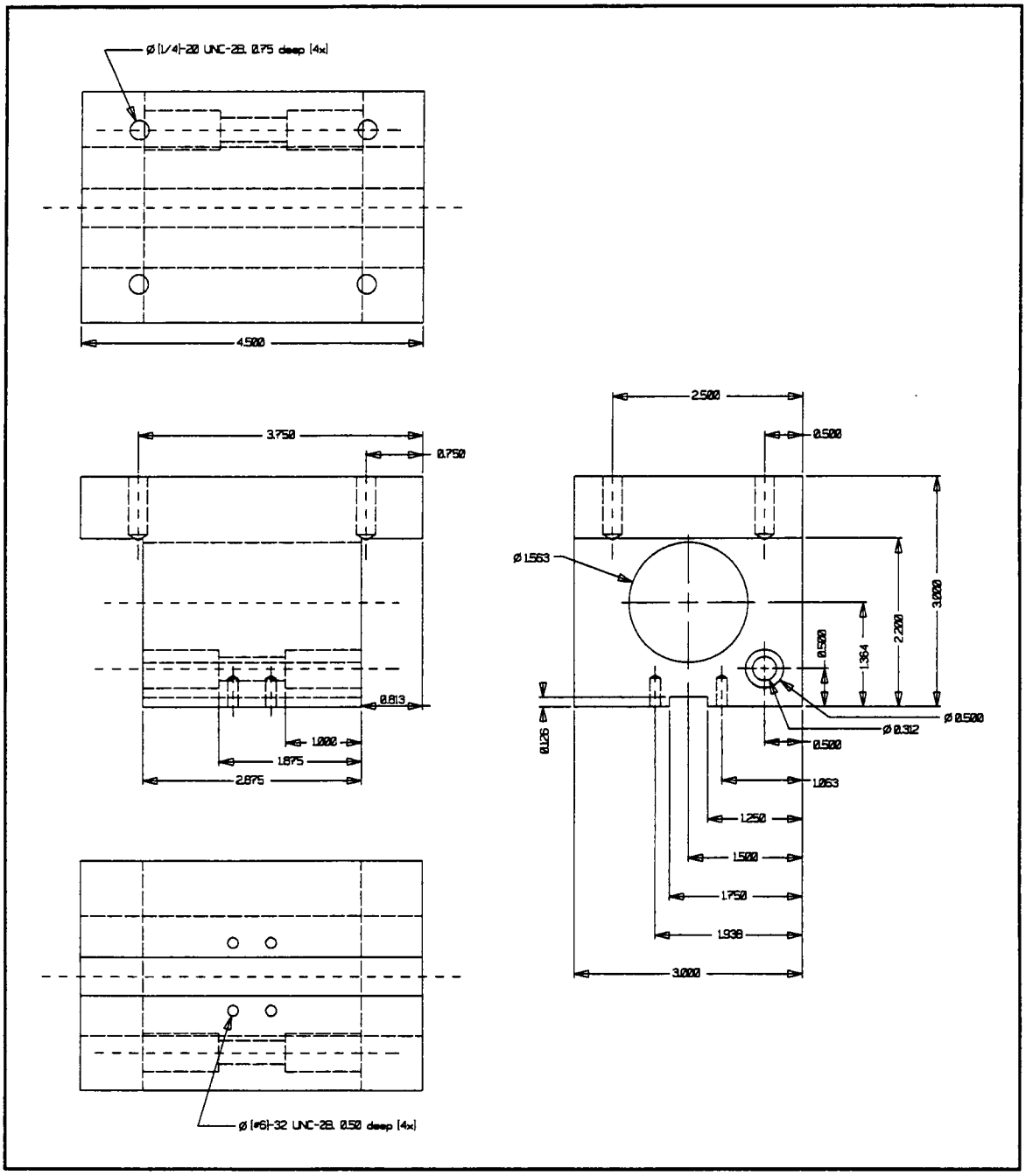



**Figure 2.7:** Machine drawing for mechanical slider.

mounting configuration had to be modified. This would be true for almost any motor that would be used on this system. Figure 2.8 shows the motor mounting plate with the DC motor's mounting hole pattern.

## User Interface

The user interface seen in Figure 2.9 uses another 68HC11 to keep the operator informed of the drive system's status. Information such as belt ratio, slider position and drive current may be monitored with this interface. Also, a new drive current may be set with this interface. All information is displayed on a two line LCD display. The user's controls consist of a display selector switch, a belt ratio dip switch, and a drive current dip switch. In addition to reading in multiplexed current, belt ratio and position data, the user interface also takes advantage of the 68HC11's analog to digital converter to read in the current proportional voltage $V_{Ia}$. The new drive currents are sent serial to the PWM controller via the TxD line of port D. Chapter 3 describes in greater detail the operation of the user interface as the majority of it's design is the software.

## PWM Circuit

The PWM circuit, as seen in Figure 2.10, is electrically very simple. It reads in the error signal from the | A-B | circuit. The error signal consists of a twelve bit magnitude named | A-B | and a sign bit named A>B. New drive currents from the user interface are received serially on the RxD line which is in port D. This information is used by the software, described in Chapter 3, to generate two constant duty cycle, pulse width

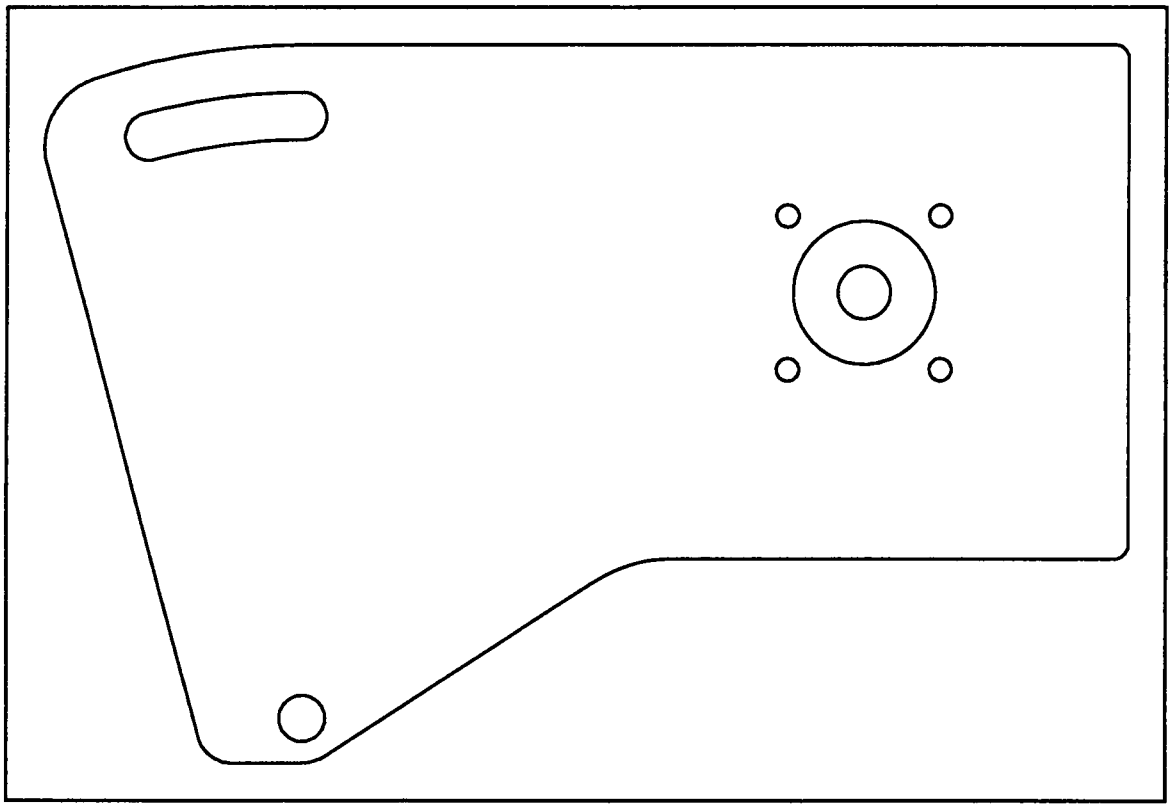


**Figure 2.8:** Motor mounting plate.

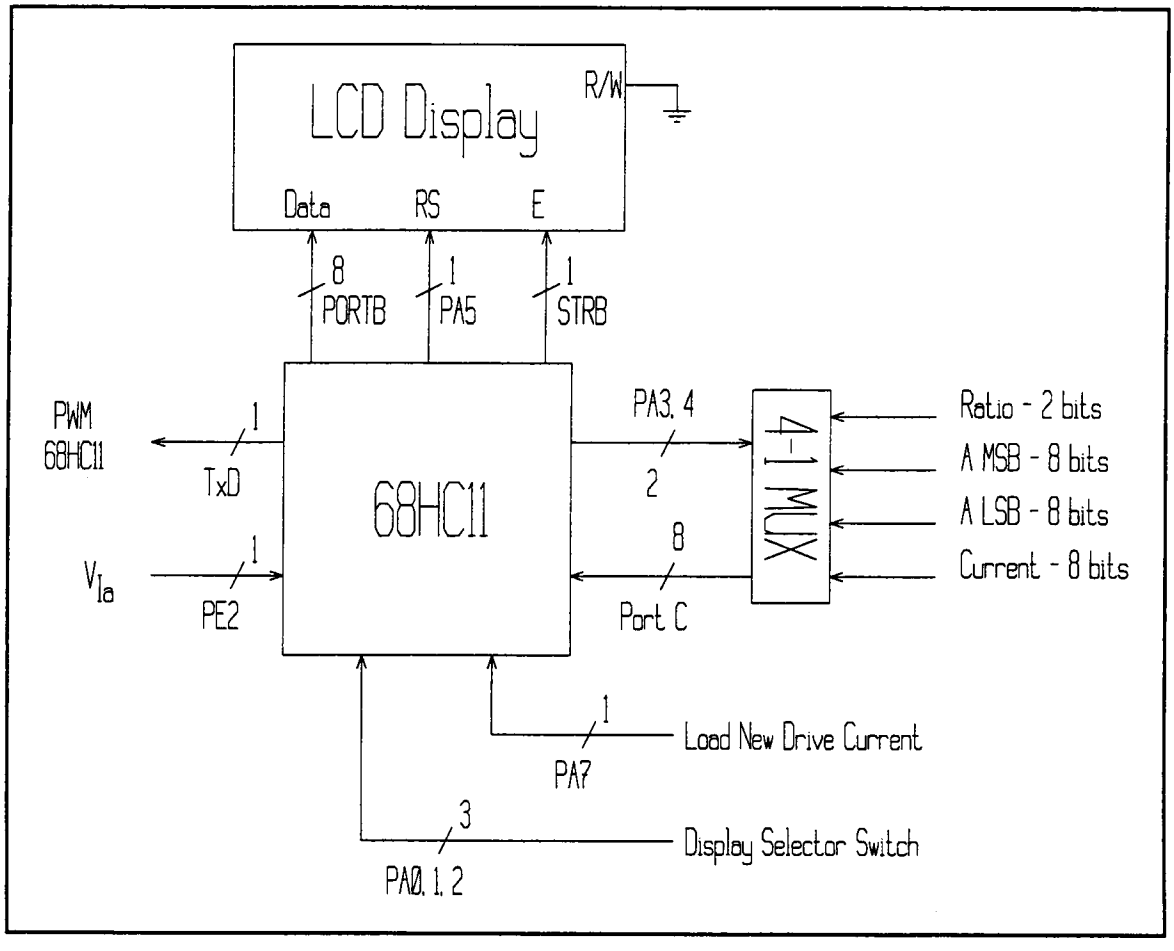



**Figure 2.9:** User interface with LCD display to show status readout.

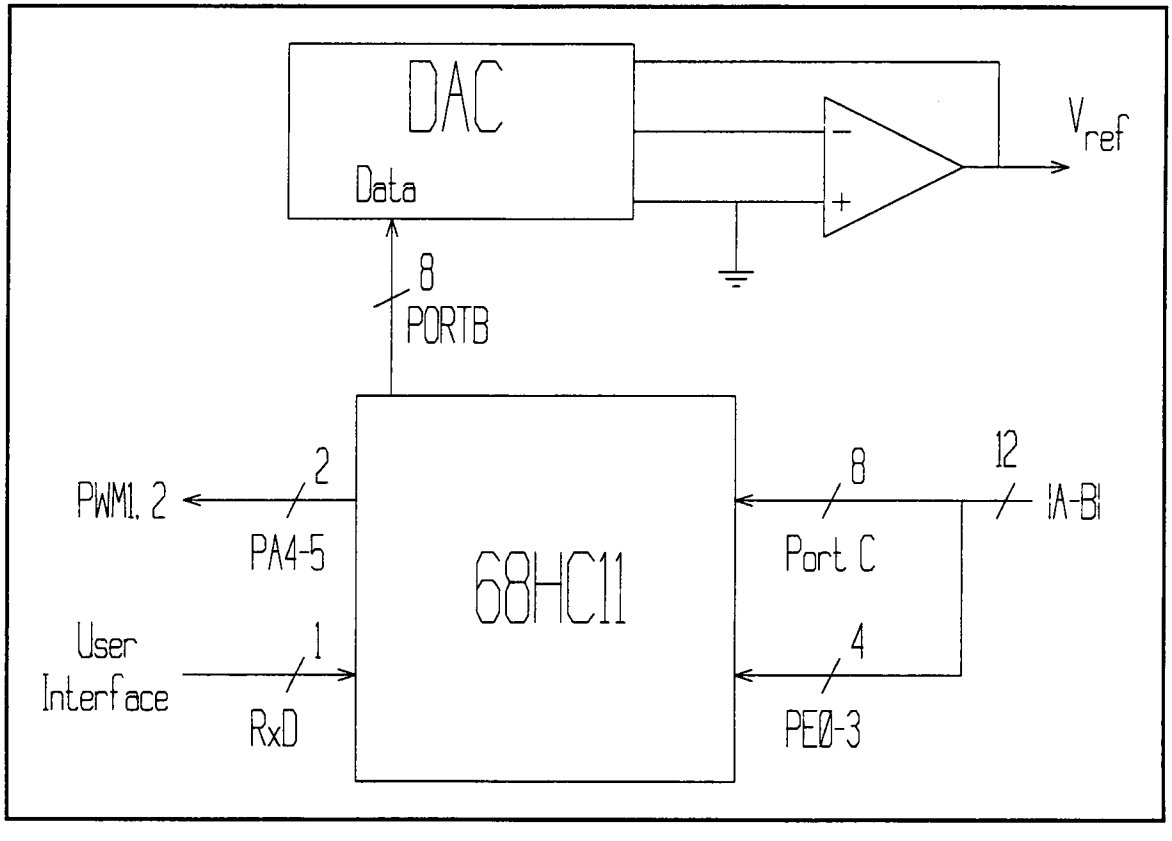



**Figure 2.10:** Pulse width modulation feedback controller.

modulation (PWM) signals and a reference voltage. Only one of the PWM outputs is used at a time, depending on whether the motor is going forward or reverse. A>B determines which PWM line is active. The reference voltage is generated by a digital to analog converter (DAC) and an op-amp.

## Drive Electronics

The drive electronics are made up of three parts, a mosfet implementation of an H-Drive, a current chopper, and protection circuitry. Figures 2.11 and 2.12 shows all three parts of the drive electronics. Drive current is directed through the motor when either PWM1 or PWM2 is high. Only one of the PWM signals is high at any given time, because if both where high at the same time the H-drive would be a short to ground. In the H-drive, the mosfet transistors are driven by a IR2110 which is a high voltage level shifting chip; International Rectifier [6]. Normally the two transistors on the high side of an H-Drive are P-channel because they don't require a gate voltage higher than the source to be energized. By using the IR2110, all four transistors in the H-Drive can be N-channel MOSFETS. The IR2110 can do this by charging a capacitor through a diode from a low voltage source when the PWM input is low and then shifting this charge to a high state when the PWM input goes high. Bridges up to 500 volts can be driven in this manner with the IR2110 chip. Figure 2.13 is a detailed pin-out and sample hook-up of the IR2110 chip.

The IR2110 chip is also used in the current chopper portion of the drive electronics. In addition to shifting TTL signals into high voltage control signals, the IR2110 has an

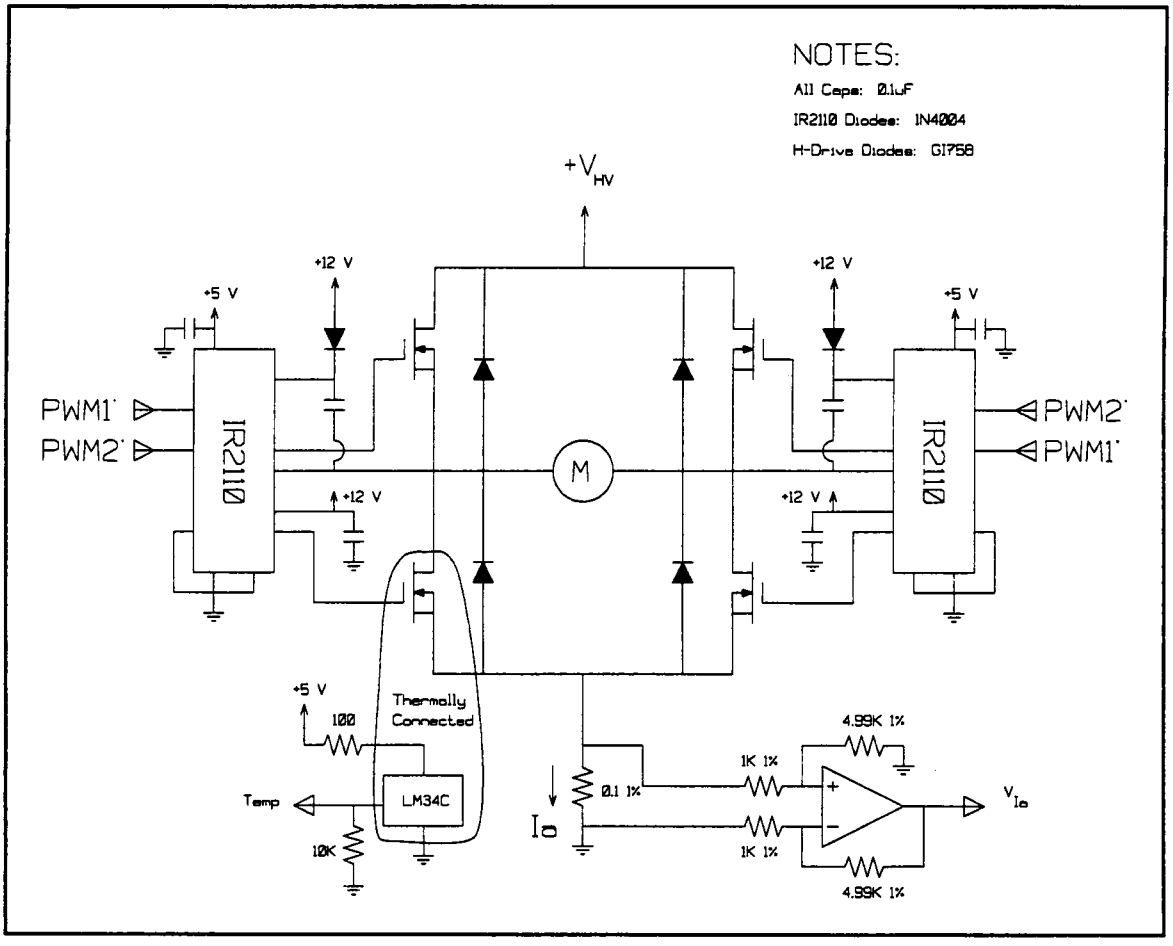



**Figure 2.11:** H-Drive sub-system with IR2110 drive chips.

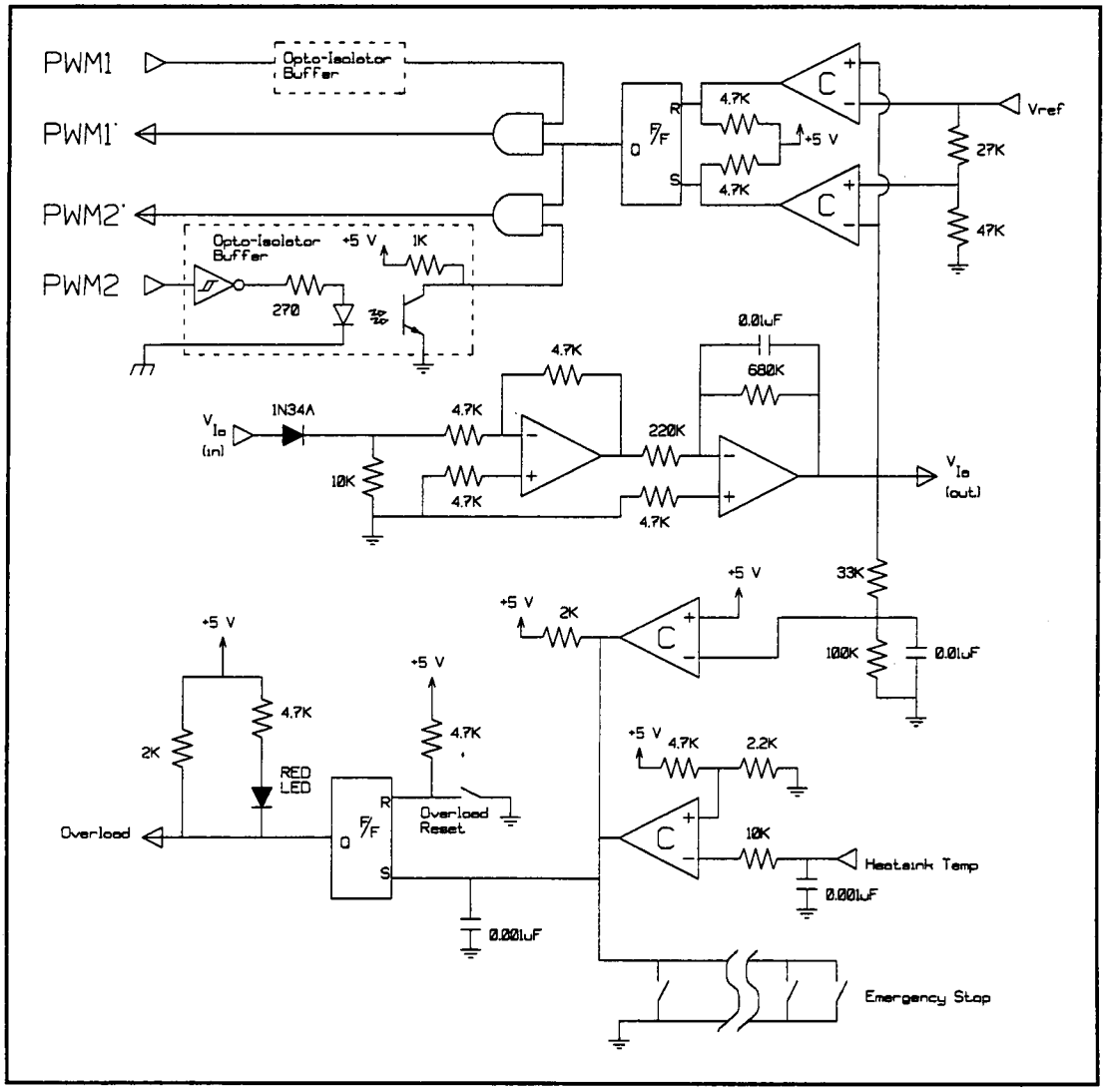



**Figure 2.12:** H-Drive sub-system support electronics.

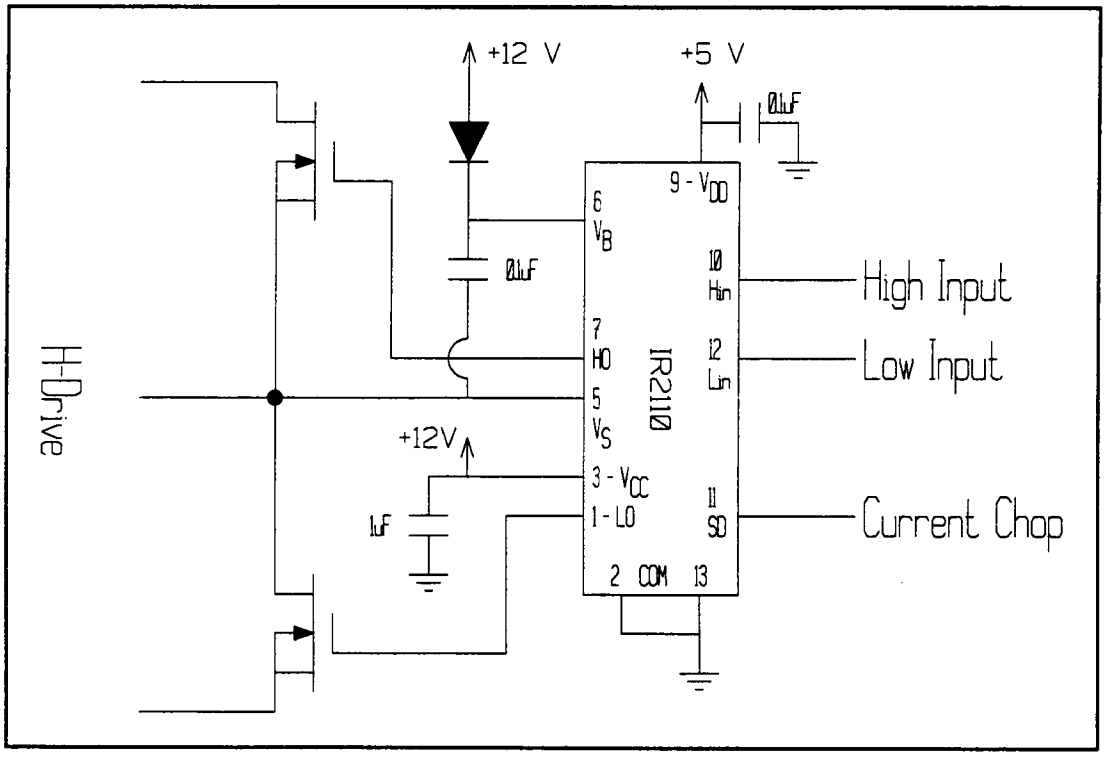



**Figure 2.13:** Detailed close-up of the IR2110 level shifting chip.

over-current cutout input, pin SD on the chip, that can be used to implement a current chopper. The equation for power dissipated is given by Equation 2.5, and is 27 watts for this application, which is below the 125 watts maximum set by the manufacturer. Current

$$Pd=R_{on}*Ia*d+(T_{on}+T_{off})*1/4*HV*Ia*freq \quad (2.5)$$

chopping is used to limit the PWM input on a pulse per pulse basis. This happens because the output of the IR2110 is only high as long as the current pin SD is low. When SD goes high, both it and the input must reset to low before the output responds to another high input. Because of this, the PWM input and the current feedback loop are not completely independent.

The third part of the drive electronics is the linear section that generates a voltage proportional to the drive current and two shutdown signals. A 0.1 ohm sense resistor between the H-drive and ground is used to measure the drive current. Before comparing can be done, the voltage across the sense resistor is amplified to a useful level. This amplified voltage is compared to a variable voltage from the user interface that allows the drive current to be adjusted to meet the users requirements. If the amplified voltage exceeds a fixed safety set point, an overload signal is sent to the power supply section. The thermal cut out works along the same principal except it's input voltage is generated by a temperature sensor with an output of 10 mV per degree fahrenheit above 0° F.

## Safety System

The safety circuit in Figure 2.14 has two main safety sections. User protection is provided by a GFI (ground fault circuit interrupter) outlet. A GFI outlet disconnects the

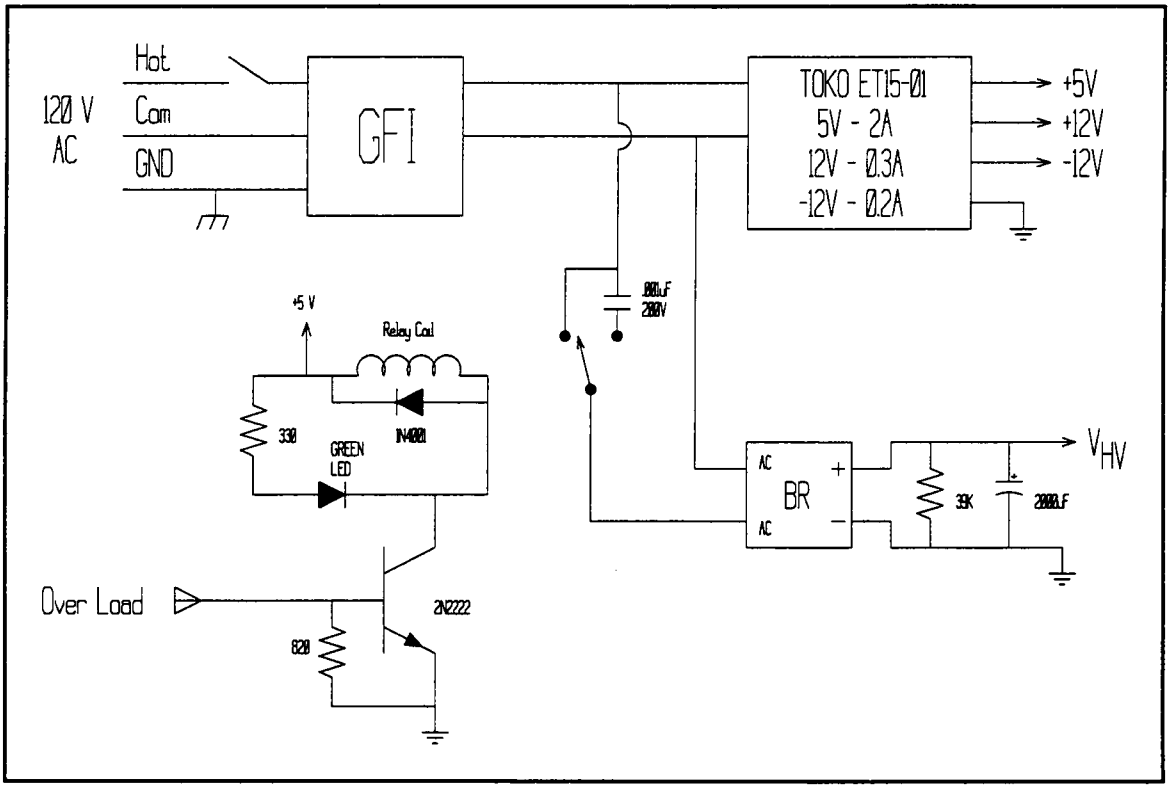



**Figure 2.14:** Safety system and power supplies.

circuit if the amount of current in the common line is different than the amount in the hot line. Section two protects both the equipment and the user. The high voltage to the drive electronics is supplied through a relay that opens on an over current, on over temperature or on an emergency stop signal. Both the over current and over temperature signals are generated by and protect the drive electronics. Emergency stop is for the user to hit in case of danger or to shut the system down to protect it.

## Motor Model

The mechanical system consisting of the weighted slider, belts and pulleys can be expressed in a mathematical model. However, most of the components of the mechanical system can be reflected back trough the belts to the motor model as seen in Figure 2.15. The system has a near constant damping friction of 0.8 in*oz. Ranges for the system's inertia can be calculated using Equations 2.6.

$$Js = Jp1 + (N1/N2)^2 * (Jc + Jp2 + Jp3 + Jpl) \qquad (2.6)$$

Table 2.3 gives a description of each of the variables used in the motor model. Because of the difficulty in calculating the friction values, they were measured using the finished mechanical system. All other parameters were either calculated or obtained from the manufacture's data sheets; Globe [7].

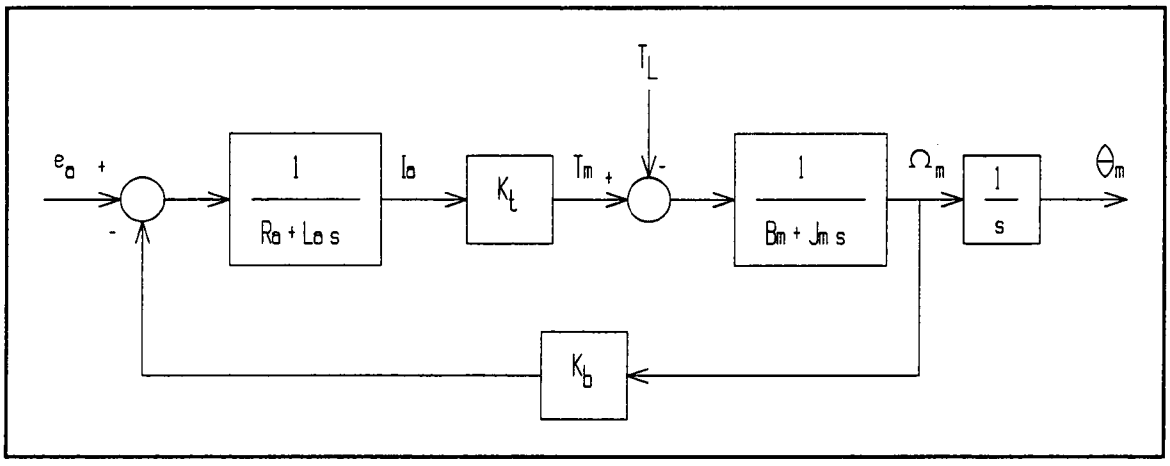



**Figure 2.15:** Block diagram of the motor.

**Table 2.3:** Motor model variable parameters.

| Var | Desc | value | units |
|---|---|---|---|
| Ra | Armature Resistance | 0.23 | ohms |
| La | Armature Inductance | 115e-6 | henrys |
| Kt | Torque Constant | 3.80 | oz-in / A |
| Bm | Damping Friction - Motor | 0.115 | oz-in-sec |
| Jm | Inertia - Motor | 2.3e-3 | oz-in-sec$^2$ |
| Kb | Back emf Constant | 2.8325 | mV / rpm |
| TL | Damping Friction - System | 7.96 | oz-in |
| Js | Inertia - Mechanical System | variable | oz-in-sec$^2$ |
| Jp1 | Inertia - pulley on motor | 4.785e-4 | oz-in-sec$^2$ |
| N1/N2 | belt ratio | 1 to 1/3 | 1 |
| Jc | Inertia - empty carriage | 2.832e-2 | oz-in-sec$^2$ |
| Jp2 | Inertia - large pulley | 7.656e-3 | oz-in-sec$^2$ |
| Jp3 | Inertia - pulley inside corner | 4.785e-4 | oz-in-sec$^2$ |
| Jpl | Inertia - payload | variable | oz-in-sec$^2$ |

# Chapter 3

# SYSTEM SOFTWARE

The DC servo drive system contains three 68HC11 microcontrollers and a dedicated hardware based math circuit; Motorola [8] and Greenfield [9]. Each of these systems required programming. All of the microcontroller circuits are interconnected to form a primitive parallel processor computer with each section having a dedicated task. The task was divided into three groups consisting of the PC interface, the user interface, and the PWM controller.

## PC Interface

The PC interface has two tasks to perform. Buffering the step and direction information from the PC is the interface's primary task. Homing the axis is the other task. Allowances for a priority structure were also made. Figure 2.3, on page 7, shows how a system with four axes is wired. After each step or home cycle the decision is made to either initiate another step or initiate a home cycle based on the input **RESET**. Figure 3.1 depicts this decision making process between the step and home subroutines. The assembly language source code for the PC interface is located in Appendix C.

Motion data from the PC comes in the form of a **STEP** command and a **DIR** command. This must be converted into two asserted low signals named **count_up** and **count_down**. Figure 3.2 presents the flow chart for the step subroutine. If the PC is not currently sending a **STEP** command, the subroutine is bypassed. When a **STEP**

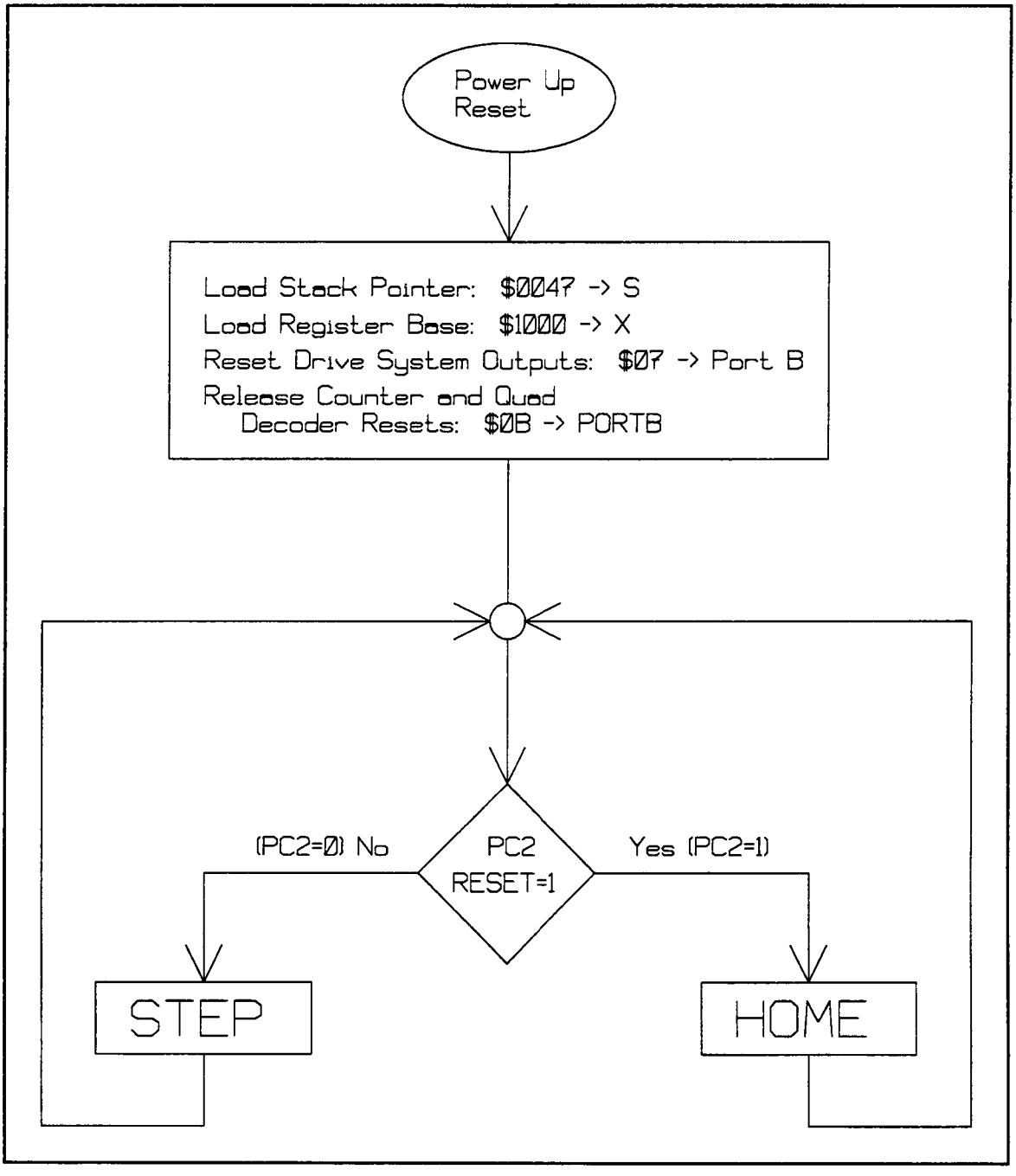



**Figure 3.1:** Flow diagram of the main section of the PC interface.

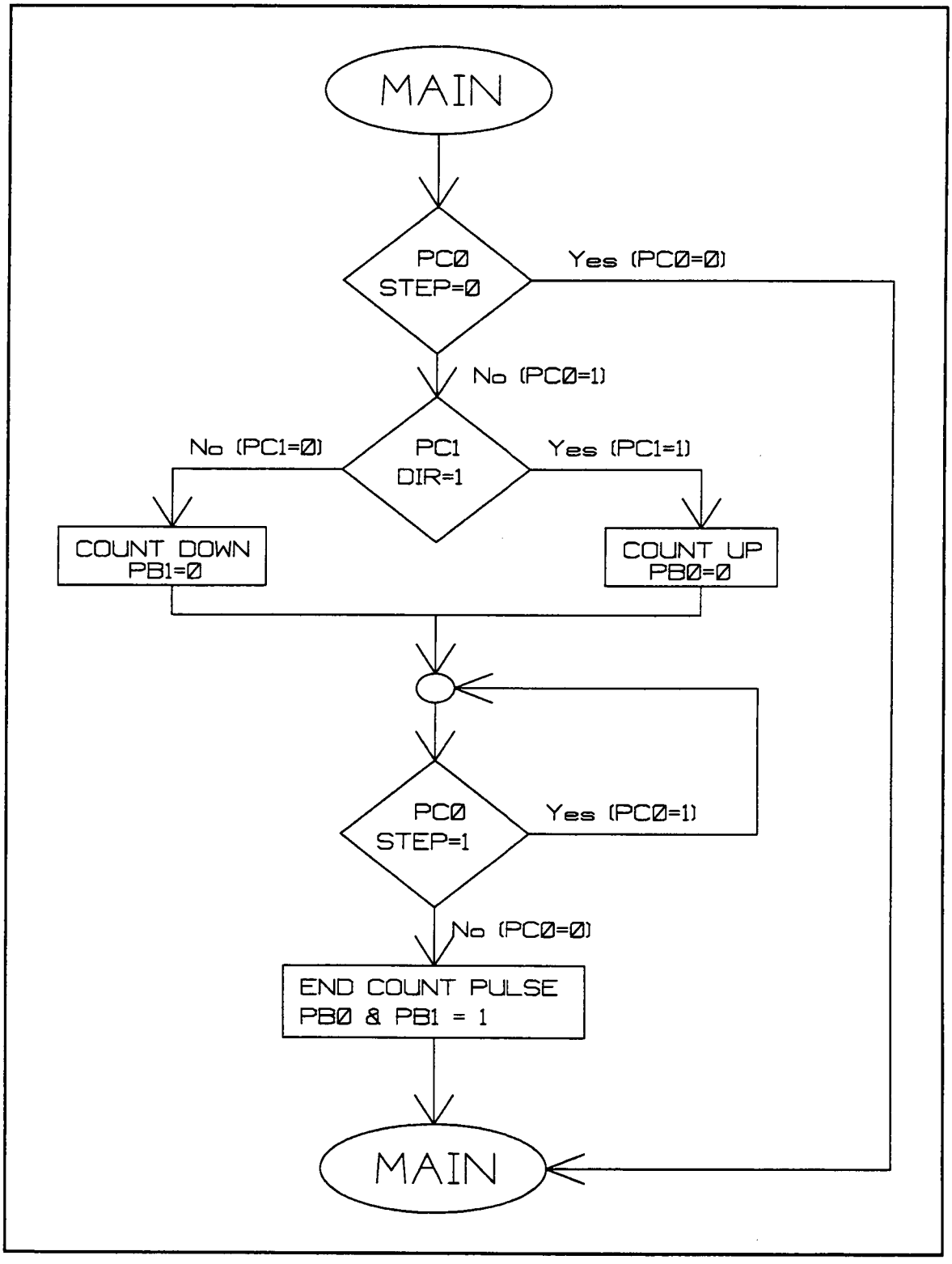



**Figure 3.2:** Flow diagram of the step buffering subroutine.

command is detected, **count_up** or **count_down** is set low depending on the **DIR** input. Both count outputs are reset high after the **STEP** input goes low.

When a **RESET** is detected in the main section of the code, the homing subroutine takes control. Figure 3.3 shows the flow of the homing subroutine. Following a reset of the counters and quadrature decoder, the axis slider is moved away from the home end of the axis toward the center of the axis travel. Home position is actually a fraction of an inch toward the center of travel from one end of the axis. If the home position where at the very end of slider travel, the slider would have to hit the corner block every time it was send to the home position. By moving the home position out just a bit, impact damage is avoided. The reason for stepping out 50 steps before going back to home is to ensure that the axis is not already past the home position. If it were past the home position, the slider would be driven into the corner block with no possibility of getting to the home position.

The priority structure is also controlled by the homing subroutine. When the axis has been moved to the home position, the axis ready (**AR**) output goes high. The **AR** of one drive axis can be fed into the next drive axis's **RESET** input. **AR** stays high until the input reset goes low. This causes the axis to home in a daisy chain fashion until the last axis in the chain sends it's axis ready signal high. The last **AR** is read in by the PC. Upon receipt of an **AR**, the PC releases it's reset. After the release of the reset has propagated through the all the drive systems, they are ready to receive step commands again. A delay time of 40 μSec should be anticipated for each drive in the chain.

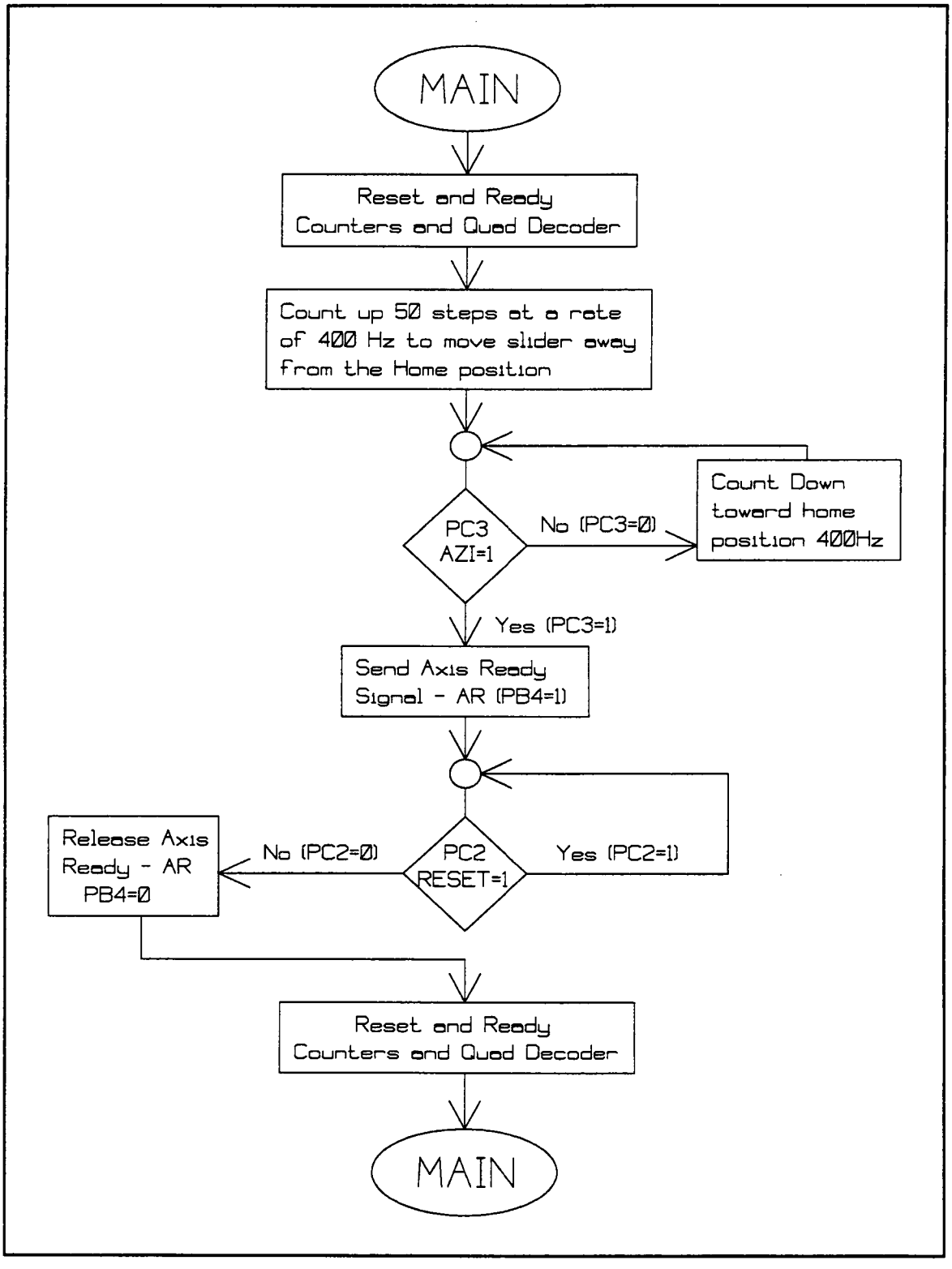



**Figure 3.3:** Flow diagram of the automatic homing subroutine.

## | A-B | EPROM Programming

The | A-B | circuit as described in Chapter 2 is made up of TTL chips and three identical EPROMs. While EPROMs are physically pieces of hardware, their contents are called firmware. A C program, which is included in Appendix C, was used to generate the EPROM's contents by using Equations 3.1 through 3.6.

$$a = A - (A>B) * C_{in} \tag{3.1}$$

$$b = B - (1 - A>B) * C_{in} \tag{3.2}$$

$$\text{if } (A>B = 1 \ \& \ a - b < 0) \text{ or } (A>B = 0 \ \& \ b - a < 0) \text{ then } C_{out} = 1 \tag{3.3}$$

$$a = a + 16 * (A>B) * C_{out} \tag{3.4}$$

$$b = b + 16 * (1 - A>B) * C_{out} \tag{3.5}$$

$$Q = abs( a - b) \tag{3.6}$$

Given the two 4 bit values **A** and **B**, a carry in bit and the information of which is larger, **A** or **B**, the EPROM generates two outputs. The first output named **Q** is the 4 bit section of the absolute value of difference between **A** and **B**. A carry out is also generated so several EPROMs can be chained together to compare numbers greater than 4 bits. The two outputs are merged together into one byte of data using equations 3.7.

$$Out = Q * 16 + C_{out} * 8 \tag{3.7}$$

The data generated by the C program for the EPROM is stored in standard Intel HEX file format which contains more that just the raw EPROM data. A line from a formatted file starts with an introductory leader consisting of ":10", the line address, the data and a

check sum byte. The check sum byte is generated from the data and the data's address in the eprom. Equation 3.8 shows the calculation used to generate the checksum.

$$\mathbf{chksum} = [0 - (\sum_{n=0}^{n=15} \mathbf{Out_n} + \mathbf{Addr_{LSB}} + \mathbf{Addr_{MSB}} + 16)] \;\&\; \mathbf{0xFF} \qquad (3.8)$$

## User Interface

The user interface is the largest and most complicated of the three microcontroller programs. A listing of the program for the user interface can be found in Appendix C. Upon power up, the LCD and serial ports are initialized. As can be seen in Figure 3.4, the main section of the user interface program reads the input from the selector switch and sends program control to the indicated subroutine. There are four main subroutines, one to display axis position, one to display motor current, one to set desired motor current, and one to set the belt ratio in the user interface program.

The position subroutine can display either the physical axis position or the desired axis position by physically moving the 16 bit data cable from the output of the axis decoder's counter to the output of the PC interface's counter. Figure 3.5 shows what steps are executed in displaying the positional data. First, two text messages are printed on the LCD display. Some data manipulation is necessary to get the data ready to be printed one character at a time. Finally the units message is printed. After a 0.1 second delay the routine is either repeated or control is returned to the main subroutine depending on the position of the selector switch.

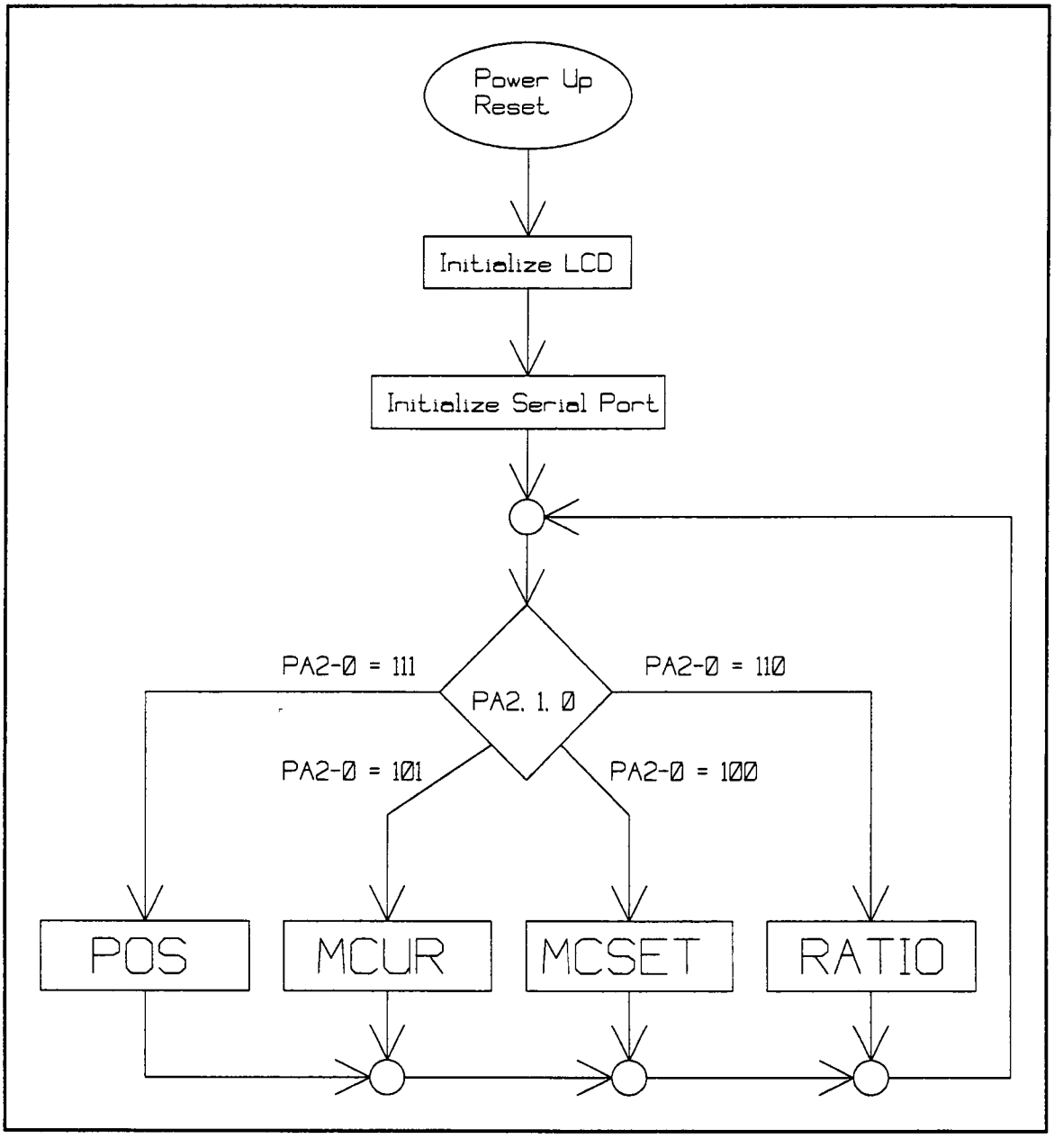



**Figure 3.4:** Flow diagram of the main section of the user interface.

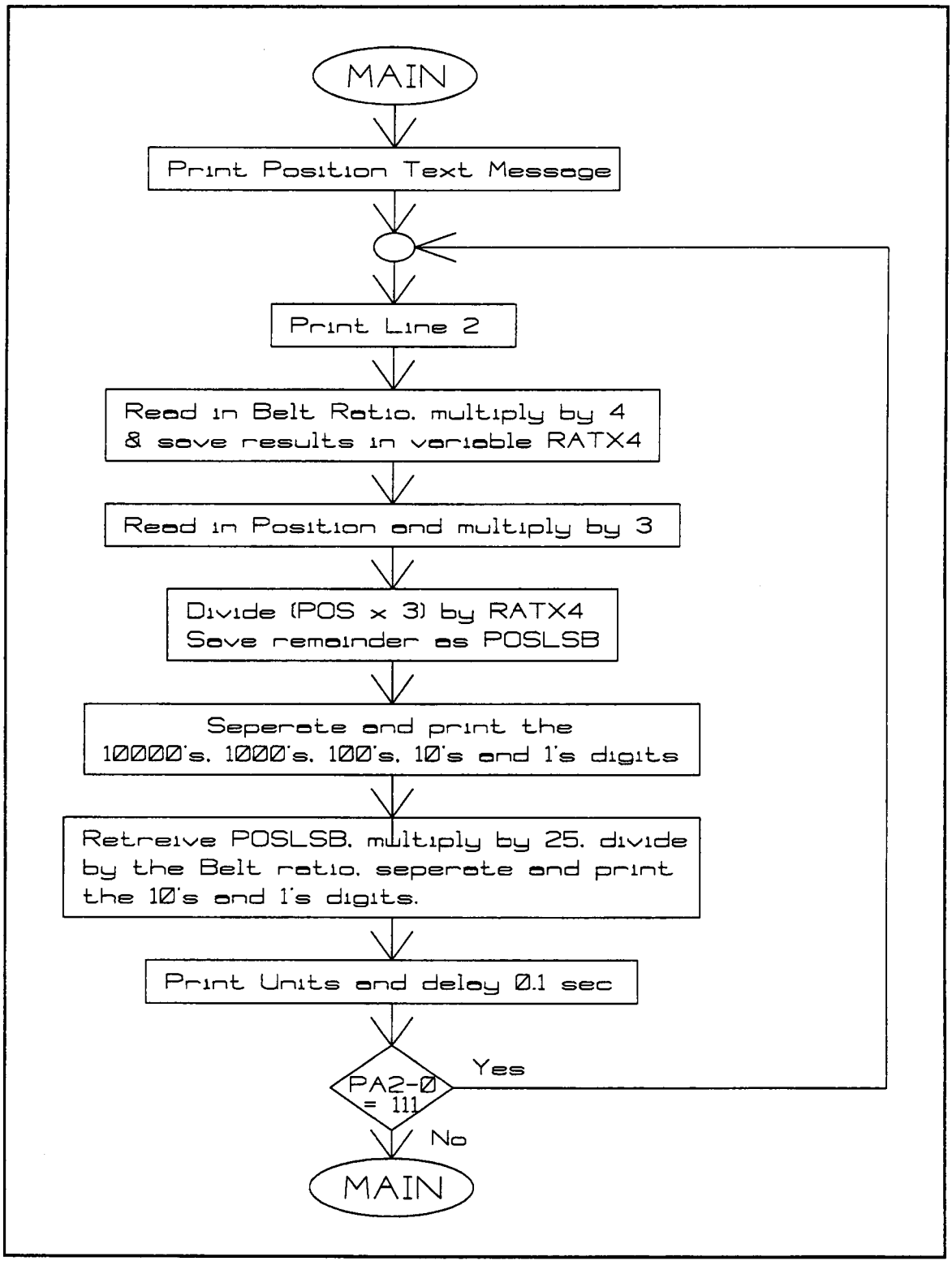



**Figure 3.5:** Flow diagram of the subroutine for position display.

The motor current flow chart seen in Figure 3.6 starts off the same way as the position subroutine with the printing of two text messages to the LCD display. However, in this routine the data comes from the A/D converter instead of from a digital port. Again the data is manipulated and then printed one character at a time to the display. Then a units message is printed. After a 0.1 second delay the routine is either repeated or control is returned to the main subroutine depending on the selector switch.

The subroutine used to set the desired motor current starts off much as the last two subroutines, except the data is read in from an eight position dip switch. After printing the units message and delaying for 0.1 seconds, the program control either returns to the main subroutine, loops again, or serially transmits the new current data to the PWM controller depending on the selector switch. Figure 3.7 shows the current set-point flow chart. While the serial data transfer is occurring, a text message is displayed. When the data transfer is complete, a new message is displayed that instructs the user to move the selector switch to another position.

Figure 3.8 shows an example of an important subroutine used by all three of the display subroutines discussed so far. This subroutine takes a large data word and prints out the digit characters, one at a time. Data to be printed is placed in register D and the digit to be separated is placed in register X. After an integer divide the quotient goes into X and the remainder goes into D. The contents of X and D are then exchanged. This places that raw number to be printed into D. If hex 30 is added to a raw number, the result is the ACSII value that corresponds to the original number. This new ACSII number is then written out to the LCD display. After a 40 µSec delay to allow the LCD

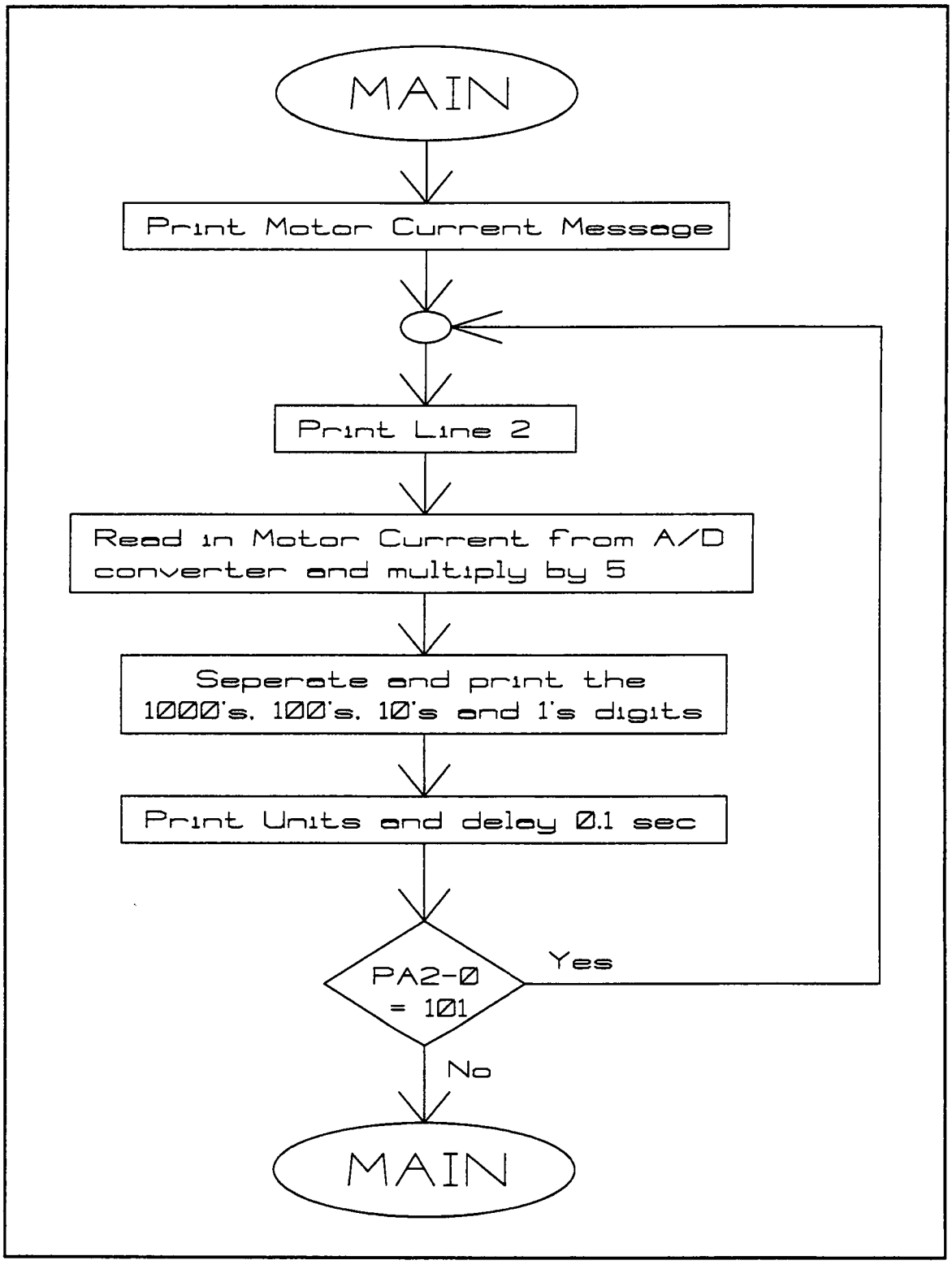



**Figure 3.6:** Flow diagram of the subroutine for displaying motor current.

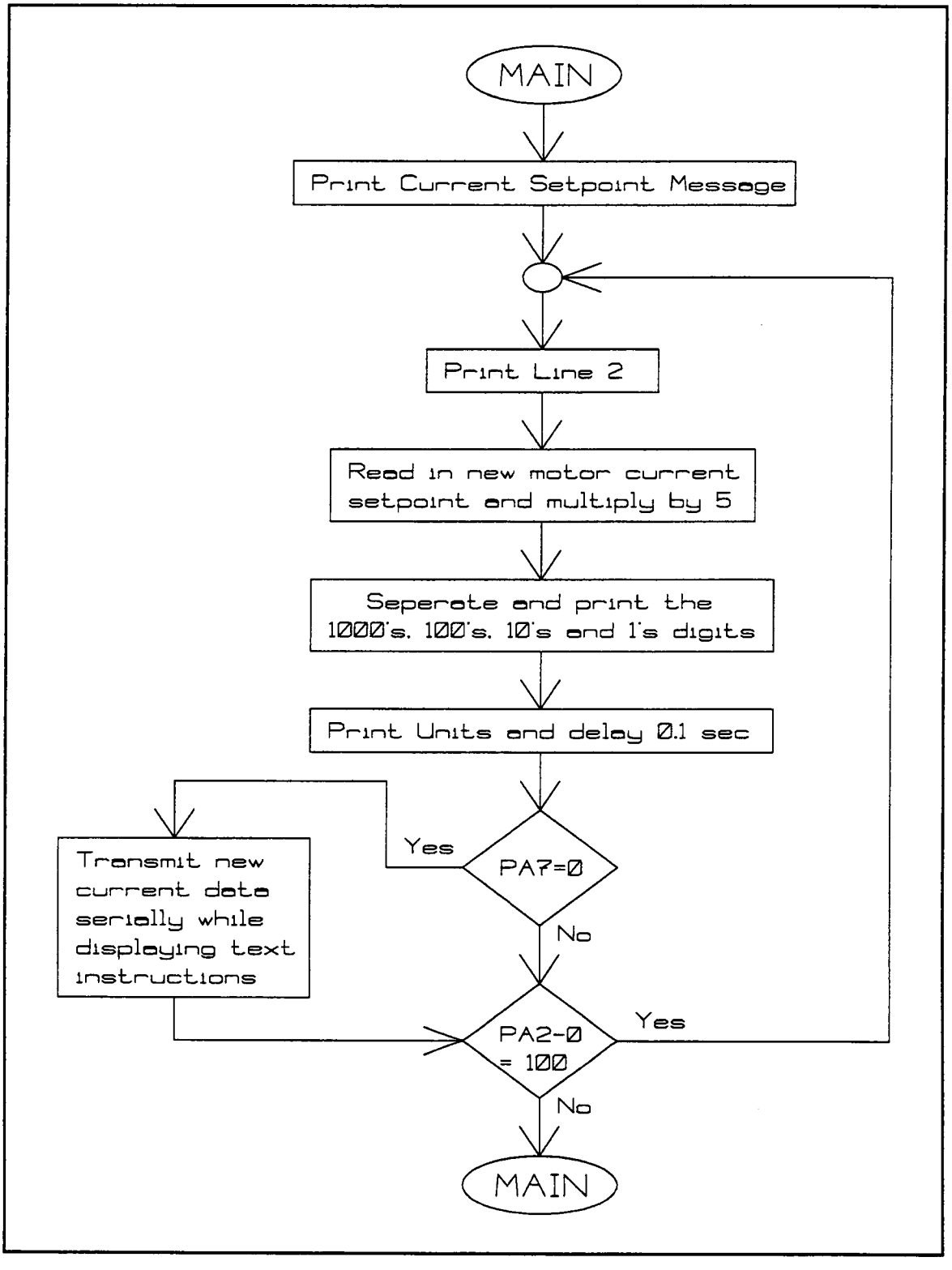



**Figure 3.7:** Flow diagram of the subroutine for setting desired motor current.

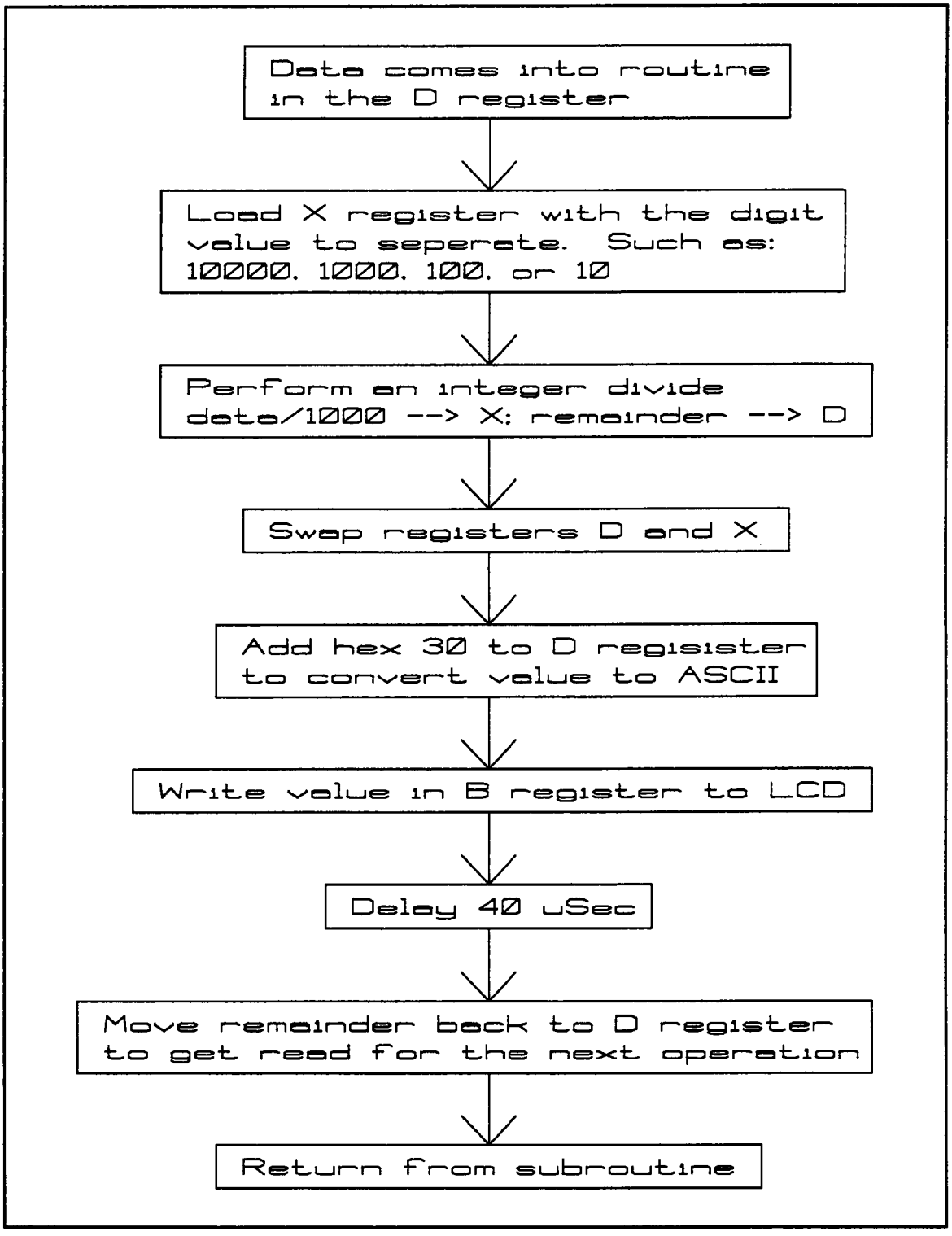



**Figure 3.8:** Flow diagram of the subroutine for separating and printing one digit of a large number.

time to complete the transfer, the remainder is moved back to D. The subroutine is ready to be repeated after the next lower power of ten is loaded into the X register.

The belt ratio subroutine is the last of the four major subroutines in the user interface. This routine starts by printing one text message and moving to the second text line on the LCD. What is printed for the second text message depends on what is read in for a belt ratio. Having four predetermined text messages for this subroutine was easier than having to perform calculations to print the appropriate resolution for a given ratio. Just as with the other three subroutines, a delay of 0.1 sec is followed by a check of the selector switch to see if the routine is repeated or if control reverts back to the main subroutine. Figure 3.9 details the operation of the user interface.

## PWM

The last program needed to control the motor is the PWM controller program. A flow chart for the PWM controller can be seen in Figure 3.10. As this program uses the serial port to receive data, the serial port must first be initialized with baud rate and data format information before the program execution can move on to the control loop. This program employs no subroutines and the control loop executes endlessly.

The first action in the control loop is to see if any new serial motor current data has been received. If new data has been received, it is stored in a variable named **Vref**. Regardless of new data or not, the next task is to turn on either the **PWM1** or **PWM2** output to the motor driver. **PWM1** causes forward shaft rotation and **PWM2** causes reverse shaft rotation. The decision to go forward or reverse is made depending on

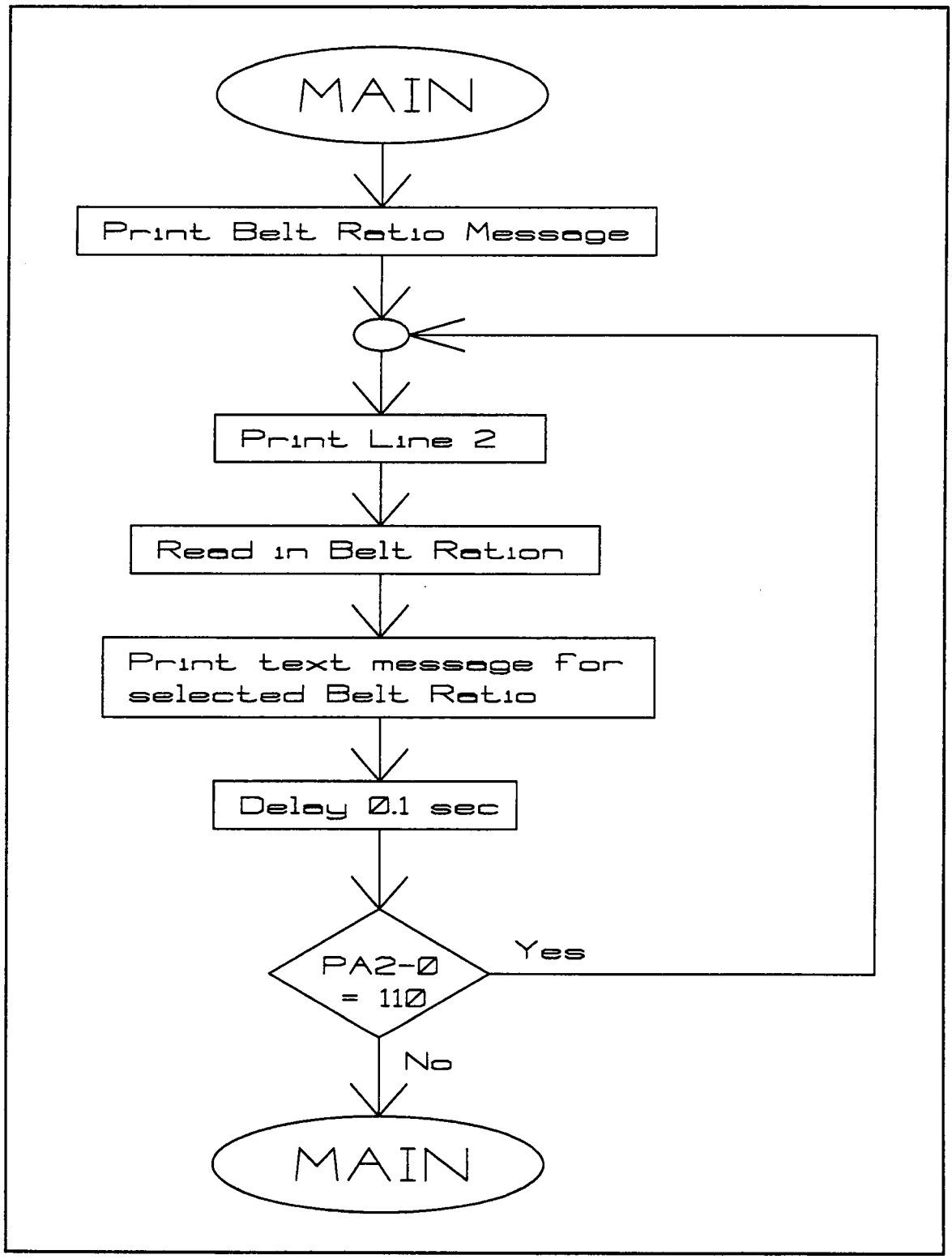



**Figure 3.9:** Flow diagram of the subroutine for displaying belt ratio.

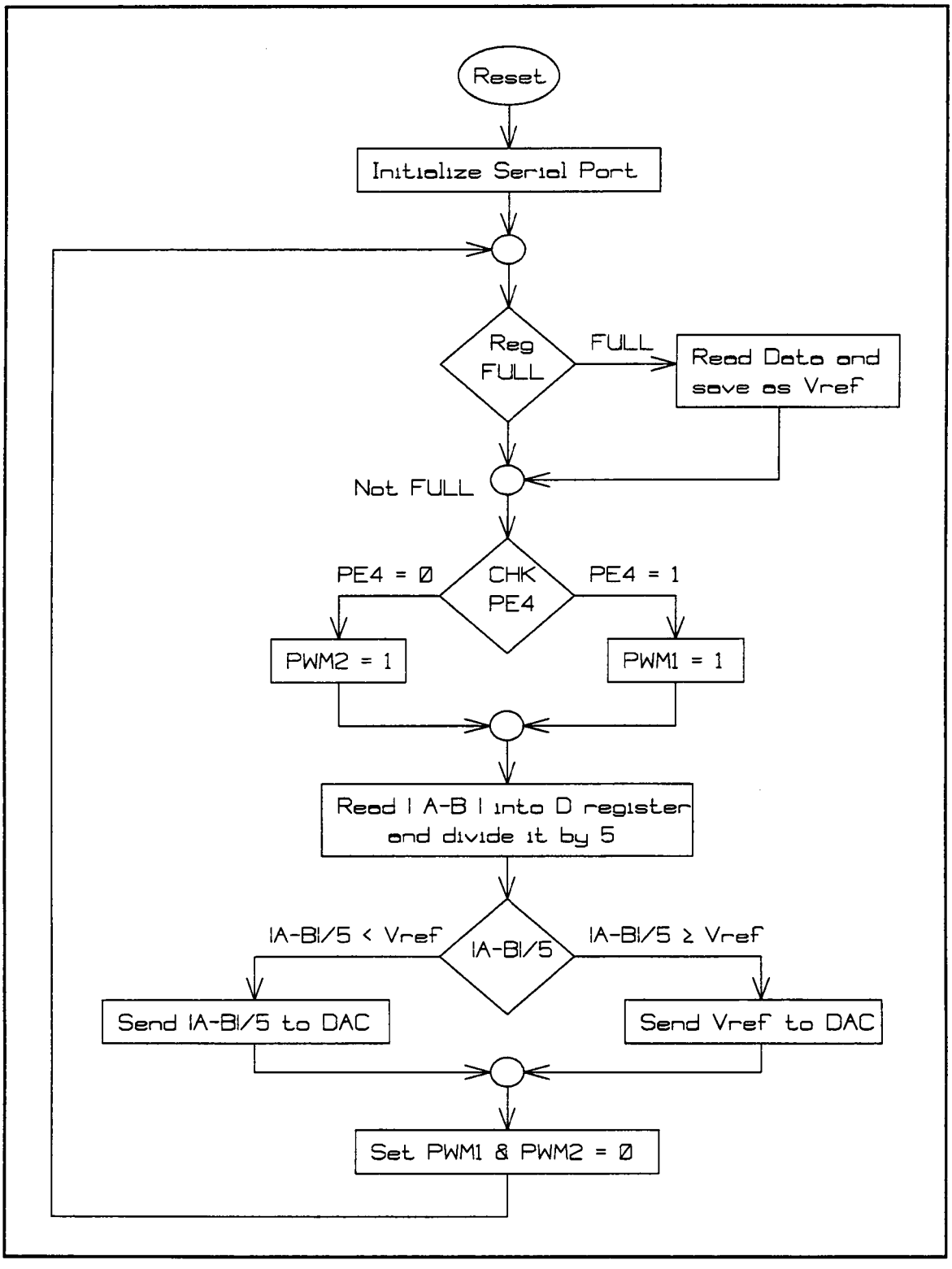



**Figure 3.10:** Flow diagram for the pulse width modulation controller program.

whether the input **A>B** is high or low. Next the magnitude of the error, **|A-B|**, is divided by five and compared to the value stored in **Vref**. If the value in **Vref** is greater then the error divided by five, the error divided by five is sent to the DAC, otherwise the value in **Vref** is sent to the DAC. The output of the DAC is used by the motor driver circuit to control the armature current. Finally both **PWM1** and **PWM2** are set to zero and the control loop repeats. The program listing for the PWM controller can be seen in Appendix C.

# Chapter 4

# SYSTEM INTEGRATION

Chapters 2 and 3 were limited to the construction and testing of the individual subsystems; whereas, this chapter explains how the individual circuits and software codes are combined to create a single drive system. Figure 2.1 and Table 2.1 (pages 6 and 7) show and explain the physical interconnections between the various subsystems.

## Final Assembly

Even though each of the subsystems was tested prior to final assembly, there were quite a few problems in getting them to work together. EMI and RFI noise was the greatest problem encountered during the final assembly. The system contains two primary noise generators a high energy pulse amplifier and a DC torque motor. Initially, they were generating sufficient EMI and RFI noise to cause all of the control electronics to fail, including the three 68HC11 microcontrollers. In order to make the many interconnections easier, all five of the power supplies' ground wires were connected together. Each 68HC11 kit has a dedicated power supply which powers the microcontroller and it's support circuitry. As seen in Figure 2.14 (page 27) there is a low voltage supply and a high voltage supply for the drive circuits. Separating the ground lines into two groups, one for the H-drive and one for the computers and supporting electronics, eliminated sufficient noise for the microcontrollers to operate without failing. Unfortunately separating the grounds broke the return loop for control lines **PWM1**,

**PWM2, Via,** and **Vref.** An optical isolator chip was used to buffer the two digital PWM signals and the two analog reference voltages were buffered through an op-amp used in a differential instrumentation mode. Figure 4.1 shows a schematic of the interface.

Even after separating the grounds, the noise was still sufficient to keep the system from running stably. RFI from the motor's brushes was causing the counters to randomly count in the | A-B | circuit . Replacing the unshielded wire from the PC interface and quadrature decoder circuits to the | A-B | circuit counter solved this problem. During this phase a worn-out drill motor was used as predictable and repeatable source of RFI and was very helpful in charting the effectiveness of the shielding improvements.

Fixing the grounding and shielding problems solved about 90 percent of the noise problems. Unfortunately, the H-drive was generating enough noise to cause spurious operation of it's own support circuitry. A variety of high frequency capacitors placed on each of the integrated circuit's power leads and across the high voltage power supply was sufficient to filter out the last of the system's de-stabilizing noise.

## Controls System Analysis

The original controller design was very simple and determined experimentally. There are two regions of control in this simple controller. In the first region, the motor current is varied linearly with the error signal | A-B | up to an upper limit at which point it is in the second region where the motor current stays constant at the maximum set by the user interface. The range of error over which the current varies linearly is zero to 63 steps which corresponds to a zero to 12.75 amp range in drive current. After the encoder was

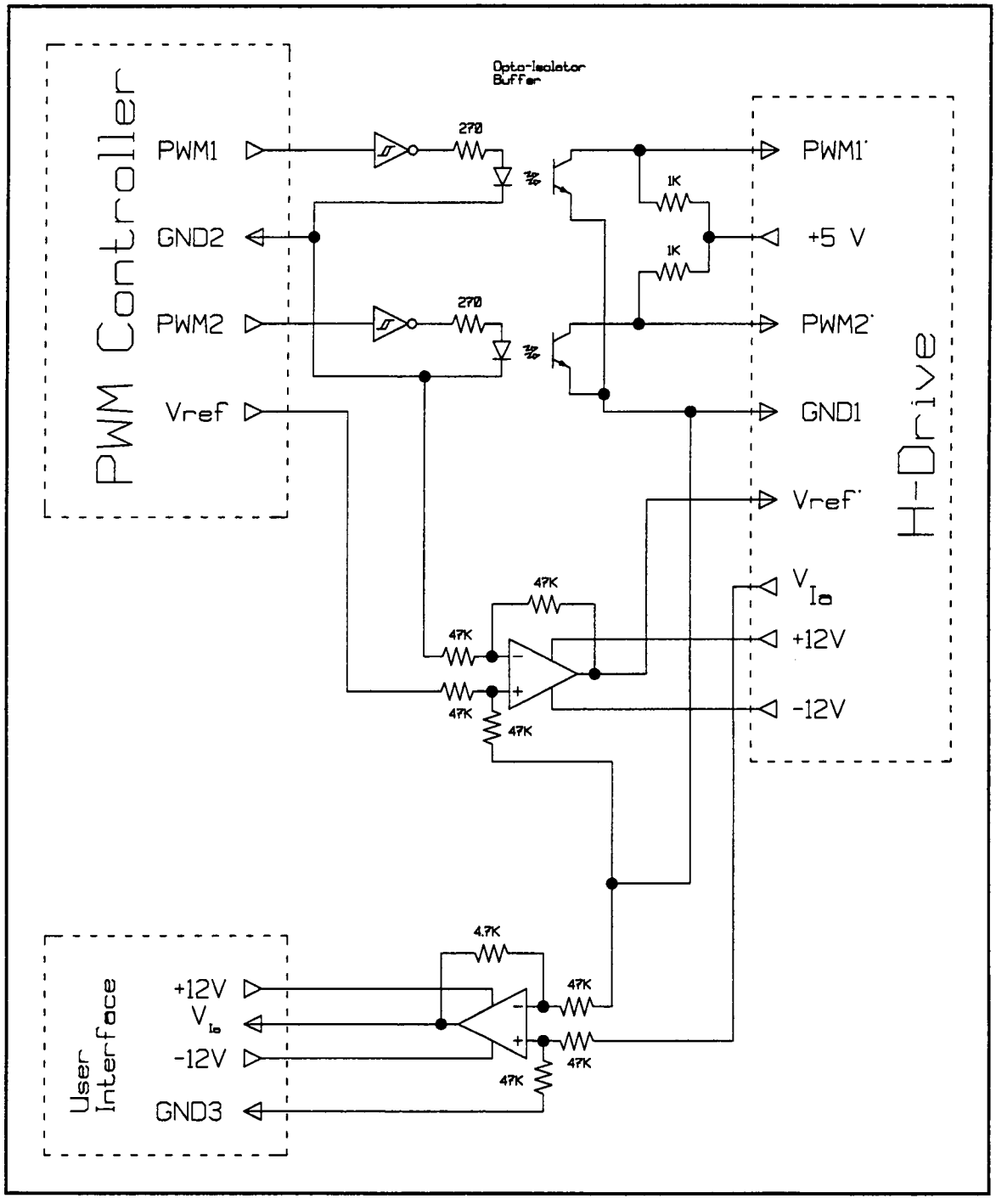



**Figure 4.1:** Schematic of opto-isolator and op-amp buffer.

moved from the far end of the axis to the motor's shaft, this control scheme produced a stable system. Even though the system was stable it did not have very good performance characteristics. Tables 4.1, 4.2, and 4.3 shows raw over shoot data for various test velocities and drive currents for payload of 0 pounds, 5 pounds, and 12 pounds respectively. The actual weight reflected back to the motor is the payload plus 3 pounds for the empty slider. A dash entry indicates either zero overshoot or that the slider fell short of the nine inch travel. Drive currents below five amps are incapable of producing satisfactory motion for this motor and mechanical system.

Before a better controller could be designed, a model of the system had to be derived. Figure 4.2 is the block diagram of the entire system. Using a payload of 5 pounds, and a drive current of 7 amps, the system was simulated and a compensator was designed. A lead compensator was selected to reduce the overshoot and stiffen the system; Kuo [10]. Figure 4.3 shows the step response of the system both with and without the compensator. Table 4.4 shows the changes in several characteristic values for the system before and after the compensator was added. In order to implement this compensator, the PWM software was re-written to include the difference equation shown in Equation 4.1.

$$Vref(nT) = 0.37 * Vref((n-1)T) + MAGAB(nT) - MAGAB((n-1)T) \qquad (4.1)$$

The calculations for the derivation of the model, the design of the compensator as well as bode plots of the system before and after compensation are in Appendix B.

**Table 4.1:** Raw overshoot data for the original controller with no extra payload.

| Pulse Freq (Hz) | Displacement [inches] for no load | | | |
|---|---|---|---|---|
| | Ia = 5 A | Ia = 6 A | Ia = 7 A | Ia = 8 A |
| 8330 | 0.085 | 0.375 | 0.637 | 0.95 |
| 5846 | 0.11 | 0.37 | 0.738 | 1.01 |
| 4726 | 0.14 | 0.45 | 0.355 | 0.285 |
| 3800 | 0.118 | 0.27 | 0.31 | 0.33 |
| 2815 | 0.15 | 0.185 | 0.2 | 0.18 |
| 2270 | 0.115 | 0.145 | 0.14 | 0.15 |
| 2270 | 0.07 | 0.05 | 0.027 | 0.125 |
| 1400 | 0.085 | 0.015 | ---- | 0.165 |
| 1022 | 0.015 | 0.01 | ---- | 0.07 |
| 535 | ---- | 0.005 | ---- | ---- |
| 218 | ---- | ---- | ---- | ---- |
| 156 | ---- | ---- | ---- | ---- |

**Table 4.2:** Raw overshoot data for the original controller with a 5 pound payload.

| Pulse Freq (Hz) | Displacement [inches] for 5 pound payload | | | |
|---|---|---|---|---|
| | Ia = 5 A | Ia = 6 A | Ia = 7 A | Ia = 8 A |
| 8330 | 0.305 | 0.69 | 0.94 | 1.208 |
| 5846 | 0.305 | 0.635 | 1.017 | 1.376 |
| 4726 | 0.314 | 0.7 | 0.995 | 1.395 |
| 3800 | 0.31 | 0.68 | 0.970 | 0.221 |
| 2815 | 0.28 | 0.165 | 0.225 | 0.342 |
| 2270 | 0.285 | 0.195 | 0.25 | 0.2 |
| 2270 | 0.15 | 0.165 | 0.125 | 0.18 |
| 1400 | 0.125 | 0.135 | 0.08 | 0.1 |
| 1022 | 0.035 | 0.065 | ---- | ---- |
| 535 | 0.03 | ---- | ---- | ---- |
| 218 | ---- | ---- | ---- | ---- |
| 156 | ---- | ---- | ---- | ---- |

**Table 4.3:** Raw overshoot data for the original controller with a 12 pound payload.

| Pulse Freq (Hz) | Displacement [inches] for 12 pound payload | | | |
|---|---|---|---|---|
| | Ia = 5 A | Ia = 6 A | Ia = 7 A | Ia = 8 A |
| 8330 | 0.235 | 0.52 | 1.07 | 1.50 |
| 5846 | 0.288 | 0.725 | 1.07 | 1.51 |
| 4726 | 0.295 | 0.64 | 1.085 | 1.62 |
| 3800 | 0.332 | 0.355 | 1.125 | 1.57 |
| 2815 | 0.305 | 0.35 | 1.09 | 0.46 |
| 2270 | 0.32 | 0.25 | 0.275 | 0.21 |
| 2270 | 0.325 | 0.23 | 0.235 | 0.2 |
| 1400 | 0.165 | 0.15 | 0.135 | 0.09 |
| 1022 | 0.07 | 0.125 | 0.06 | 0.075 |
| 535 | 0.04 | 0.01 | ---- | ---- |
| 218 | ---- | ---- | ---- | ---- |
| 156 | ---- | ---- | ---- | ---- |

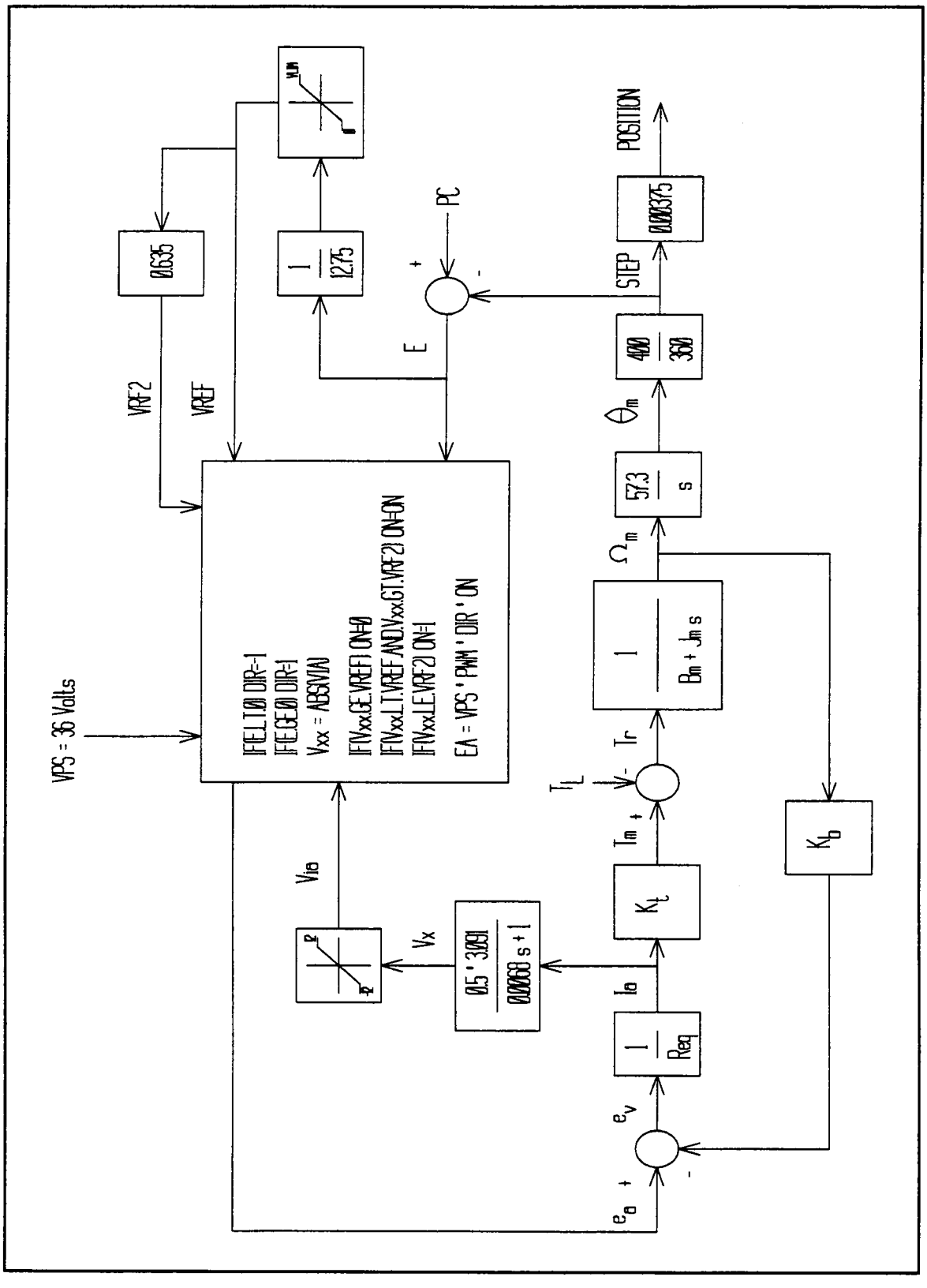



**Figure 4.2:** Block diagram of the system.

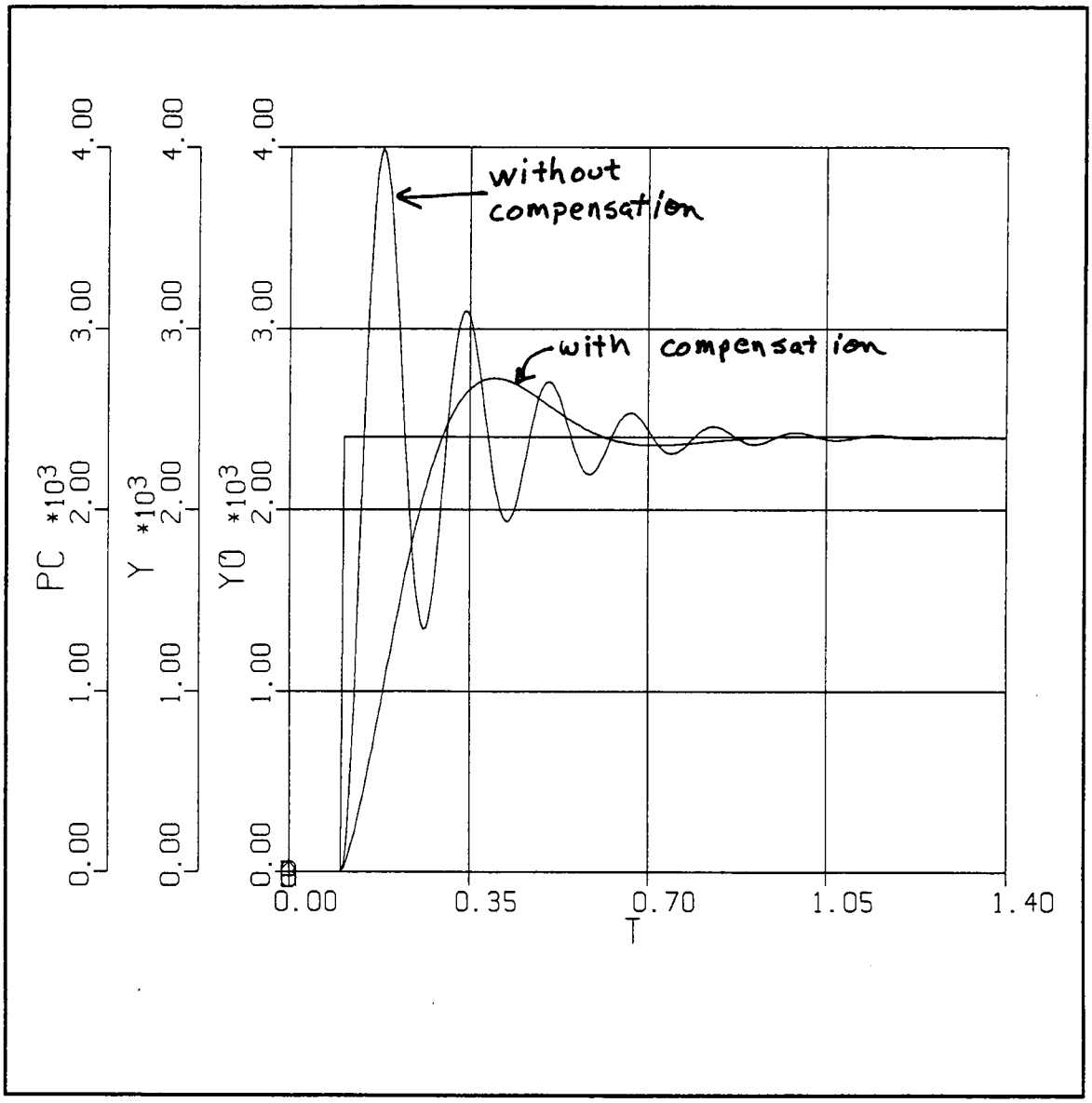



**Figure 4.3:** Step response of system with and without the compensator.

**Table 4.4:** Comparison of performance data for the system with and with out the compensator.

| Parameters | original system | system with comp |
|---|---|---|
| % Over Shoot | 66.7% | 13.64% |
| Time of peak | 0.08 | 0.30 |
| damping ratio | 0.1265 | 0.5356 |
| Phase margin | 11.8 % | 44.5 % |
| Gain crossover freq | 244 Hz | 87.6 Hz |
| Gain margin | 166 dB | 235 dB |
| Phase crossover freq | 3.46 MHz | 263 MHz |
| Mp | 16.645 | 16.7 |
| freq at Mp | 344.5 | 346.8 |
| Closed loop bandwidth | 345 Hz | 14.67 Hz |

While the compensator did improve the system's overshoot characteristics, the improvement was at the expense of the closed loop bandwidth. As with most engineering decisions, there is always a choice between conflicting performance parameters. The type of compensator that should be used with a given system depends on the customer's desired application. With a more accurate system model and more powerful analysis tools, an optimal controls solution could be found for a range of system parameters. However, such an exhaustive analysis does not fall within the original goals for this project.

# Chapter 5

# CONCLUSIONS

## Final Results

The axis drive system described in this thesis can either be used as a stand alone test stand or as part of a robotic drive system. For the current belt configuration, the system can accept pulses at a maximum rate of 8000 Hz, which corresponds to 30 in/sec. If the secondary support shaft is correctly sized, the mechanical components can easily handle a 50 pound payload. The electronics are versatile enough to operate with almost any motor that has similar or less demanding specifications. Also, with only slight modifications to the programming, the input voltage can be any readily available value from 30 volts to 180 volts. For safety considerations, all of the testing was performed at 36 volts. The effect of the motor inductance and back emf are reduced with increasing voltage.

Table 5.1 itemizes the total cost of $940 to build the drive system. Approximately half of the total cost was required for machining and assembly labor. The motor, encoder, and electronics, assembled and tested, cost $360 in small quantities. This price will easily drop to the desired $250 target if the parts are purchased in bulk quantities. A linear four quadrant power supply with a lower output power capability cost the SEDP group almost $4,000. In addition to being smaller and lighter, the switching amplifier produces less heat and is therefore more efficient than the linear supply.

**Table 5.1:** Cost summery of drive system.

| Description | Cost |
|---|---|
| Castings, Aluminum stock, and Angle iron | $100 |
| Machining of Castings | $200 |
| Machining of Slider Block | $100 |
| Machining of Heat sink | $50 |
| Thompson Bearings | $130 |
| Globe Motor | $65 |
| HP 400 count encoder | $35 |
| Custom circuit cards - 1 support electronics & 1 drive elect. | $25 (in bulk) |
| Integrated circuits components | $75 |
| Power supply components | $50 |
| Enclosure | $30 |
| Assembly labor - 8 hrs /drive system @ $10 / hr | $80 |

Even though this drive system was designed to have the same inputs as a stepping motor drive system, there are a few differences between the two types of systems. Each system has its own advantages and weaknesses. A stepping motor has a maximum overshoot of half a step, where as, the overshoot for the DC servo system can be as much as a few hundred steps. Because the stepping motor is an open-loop system, it can never correct for a missed step contrary to the capability of a DC servo system. If the stepping motor is pulsed too rapidly, the motor will vibrate. In contrast, the DC servo motor will lag behind a fast series of pulses. When the fast series of pulses either stops or slows, the DC servo motor will then 'catch-up' and lock into the correct position. The DC servo

motor does not suffer from torque drop-out like a stepping motor does. Therefore, there are no drive speeds that have to be avoided with a DC servo system.

## Recommendations

The goals established for this thesis have been met. However, there are areas that can be improved to enhance this system. Listed below are the recommended improvements.

- Decrease the size of the main shaft from one inch to 0.75 inches. As the specification for maximum payload is 50 pounds, the one inch shaft which is rated for 1000 pounds is over designed. The 0.25 inch shaft is too weak to keep the block from rotating and should be larger. A shaft size of 0.375 inches will keep the block rotating to an acceptable level for a 50 pound load in the middle of the 3 foot span.

- Condense the size of the drive support electronics. Replace the quadrature decoder circuit with a commercially available chip. Divide the drive system into two main circuit cards with the first card being the H-drive transistors, protection diodes, IR2110 drivers and sense resistor amplifier. Everything else can be placed on the second card for better noise immunity.

- Integrate all three 68HC11A1's into a single 68HC11F1 high performance microcontroller to reduce cost and improve noise immunity.

- Provide more isolation between the high current, noise generating, part of the drive and the sensitive electronics by optically isolating all of the connections between the two circuit halves including the analog signals. The analog signals can be

converted to frequency and then back to a voltage without an excessive too increase in cost. Further gains in noise immunity can be gained by placing the support electronics in one case and the H-drive amplifier in another and connecting the two using a fiber optic cable. A fiber optic cable offers the potential of improved safety.

- Write the code in such a way the controller equation can be entered in a higher language such as C or basic. This will make the system easier for someone to make field modifications to the drive system and will make it easier for students to use the system in a laboratory environment.

- Replace the speed reduction belt with a worm drive with a ratio of 8 to 1 to make better use of the DC motor speed range. Worm drive gear trains have the added benefit of being difficult to back-drive. This change is necessary if the system is to be used in vertical operations.

While this prototype is not completely ready for mass production, I believe the project is complete and has fulfilled all of my major goals stated in the original proposal which are summarized below.

- Expand the technology base for the SEDP group by providing a second motor drive system for the SEDP robot.
- Maintain software compatibility with the SEDP robot code and hardware.
- Exhibit the 68HC11 to the Smart Machinery class.
- Create a test stand that could be used in the classroom or laboratory.
- Create a drive system that is a low-cost, 'stand-alone' product capable of functioning with a variety of motors.

All of these goals have been met. Some of the more significant accomplishments of this thesis are, (a) the creation of an interface that makes a DC servo motor behave as if it were a stepping motor, (b) the design of a PWM amplifier that can work will a variety of DC motors, and (c) the development of sub-systems that can be utilized by future interested parties.

# LIST OF REFERENCES

## LIST OF REFERENCES


[1] Petrocelli, The Future of the Semiconductor, Computer, Robotics, and Telecommunications Industries, Princeton, NJ, 1984

[2] McGuire, Terri Nelson, MARKETING OPPORTUNITY ANALYSIS: The Commercialization of the SEDP Robot, MBA Intern Paper, The University of Tennessee, Knoxville, TN, May 1991

[3] Kennedy, Michael Earl. Low-Volume Production Design and Manufacture of a Low-Cost Industrial Robot, M.S. Thesis in Mechanical Engineering, The University of Tennessee, Knoxville, TN, May 1991

[4] Williams, Anthony Paul, Evaluation of a Low-Cost Industrial Robot, M. S. Thesis in Mechanical Engineering, The University of Tennessee, Knoxville, TN, May 1992

[5] Moscone, Christopher G., The Design of a Stepper Motor Torque Characterization System, M. S. Thesis in Mechanical Engineering, University of Tennessee, Knoxville, TN, May 1991

[6] International Rectifier, data sheet PD-6.011B for a IR2110 High Voltage Mos Gate Driver

[7] Globe motor company, data sheet for IM-21 Motor, revised June 1, 1990

[8] Motorola, M68HC11 Reference Manual, Prentice Hall, Englewood Cliffs, N.J. 07632, 1989

[9] Greenfield, Joseph D., The 68HC11 Microcontroller, Saunders College Publishing, 1992, ISBN: 0-03-051588-2

[10] Kuo, Benjamin C., Automatic Control Systems, 5th Edition, Prentice Hall, 1987

# Appendices

# Appendix A

Many sub-assemblies both mechanical and electrical had to be fabricated for this project. While machine drawings and circuit schematics may tell someone what is to be made they rarely give any clue as to how the part is to be made. This section explains how some of the more difficult parts were manufactured. Many of the mechanical parts are either exact copies of adaptations of the ones used to make the SEDP robot. Table A.1 gives a list of drawings from Kennedy's thesis that were helpful in making and assembling the mechanical section of this project.

**Table A.1:** Drawings from Mike Kennedy's Thesis.

| Drawing Number | Drawing Description | # of Pages |
|---|---|---|
| MEK001-DDD | Corner Block page | 3 |
| MEK012 | Right Hand Motor Mounting Plate | 1 |
| MEK021 | XY Slider Housing | 2 |
| MEK042 | Fixed Belt Clip | 1 |
| MEK043 | Adjustable Belt Clip | 1 |
| MEK046 | XY Reduction Shaft | 1 |
| ASY506-RRR | Robot Assembly - isometric view | 2 |
| ASY526 | XY Reduction Shaft with Pulley | 1 |
| KIT611-RRR | XY Reduction Kit | 2 |

## Machining - Corner Block

Both corner blocks were machined on the Bridgeport CNC milling machine. The five faces of the corner block are shown in Figure A.1. Machining of the corner block was a multi step process. Table A.2 shows the tooling needed to make the corner blocks. In each step the corner block was clamped to an 90° reference block.

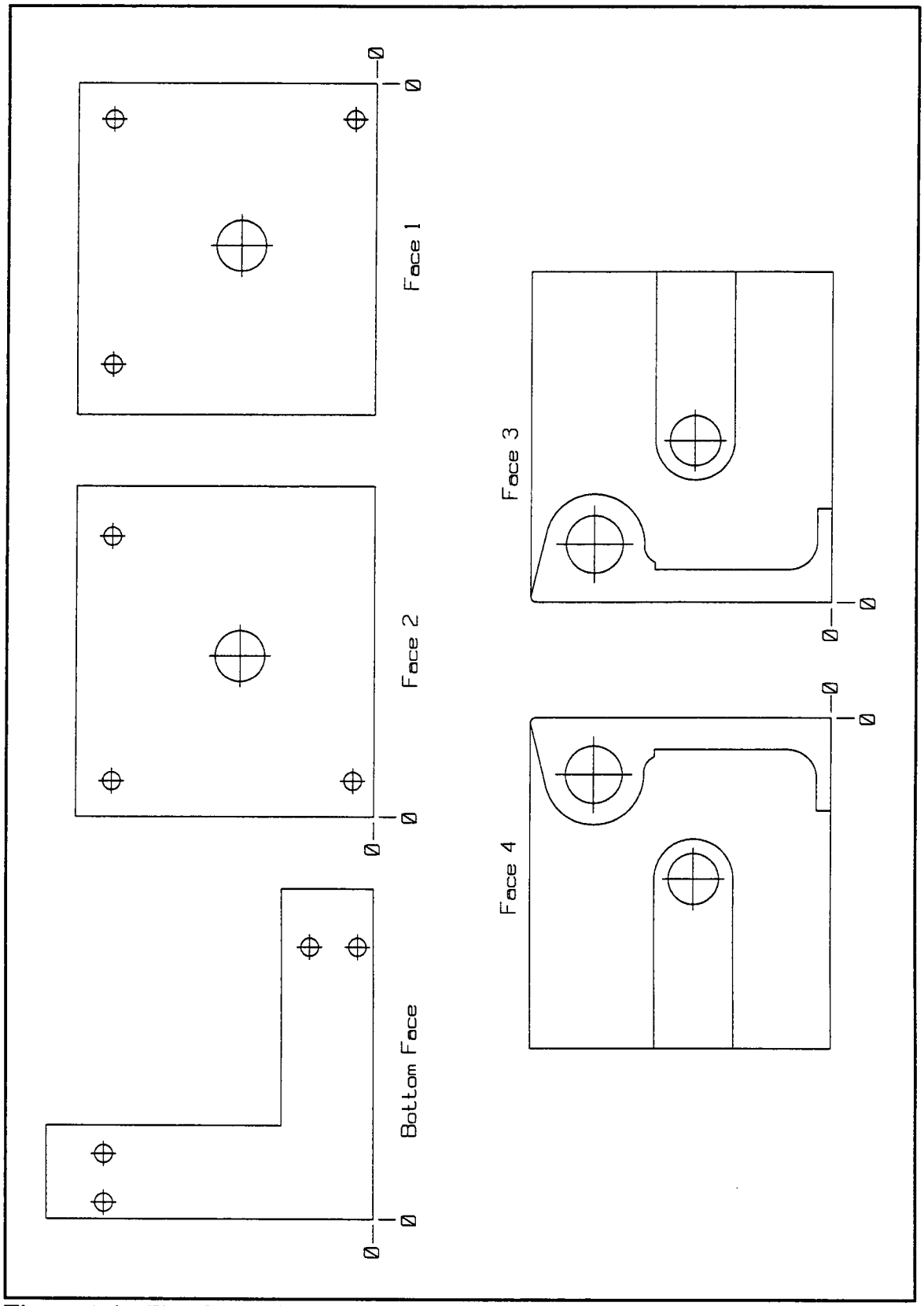



**Figure A.1:** Five faces of the corner blocks.

**Table A-2:** Machine Tools used for making the corner block

| Tool | Spindle Speed | Feed Rate | Max Cutting Depth |
|---|---|---|---|
| 4" fly cutter | 350 rpm | 1 inch/min | 0.02" / 0.005" |
| 3/4" end mill | 450 rpm | 0.5 inch/min | 0.375" |
| #G drill (0.2614") | 1350 rpm | 1.25 inch/min | n/a |
| 3/8" drill | 1200 rpm | 1.1 inch/min | n/a |
| 1/2" drill | 750 rpm | 0.75 inch/min | n/a |
| 31/32" drill | 450 rpm | 0.3 inch/min | n/a |
| 1" reamer | 100 rpm | 0.2 inch/min | n/a |
| 27/32" drill | 550 rpm | 0.5 inch/min | n/a |
| 7/8" reamer | 100 rpm | 0.4 inch/min | n/a |

Starting with face two up, the corner block was clamped in place. The face was then fly cut to remove the rough texture of the casting process. Repeated passes were needed to get the face down to flat surface. Using a 3/4" end mill face 3 was square cut to face 2.

The block was flipped and clamped into place with face one up. Again the surface was fly cut to remove the roughness left by the casting process. Face 4 was square cut to face 1 just as face 3 was to face 2 with the 3/4" end mill.

With the block reoriented with face 2 up, the square cut was repeated, this time cutting the face to it's final length. Still using the 3/4" end mill, a 1/2" deep ledge was cut along the length of the bottom face. This cut establishes a reference needed to surface the bottom. Therefore, the ledge needed to be as wide as the total amount of material to be removed from the bottom face.

The block was then clamped into place with the bottom face up. A dial indicator was used to get the ledge cut into the block level. The bottom face was then fly cut down to the level of the ledge. Four mounting holes were then drilled in the bottom. The two holes closer to the outer faces were drilled with a G drill and were later taped for a 5/16" 24 UNF thread screw. While, the other two holes were drilled out with a 3/8" drill so the screw could pass though without obstruction.

At this point the instructions vary depending on whether a **right** or **left** corner block is needed. To make a **right** corner block orient the work with face 3 up and face 2 against the reference angle. If the **left** was needed, clamp the work in place with face 4 up and face 1 against the reference. Spacers were used under the block so that the through holes needed for the bearings could be drilled without drilling into the table. The 1" shaft hole was started with a 1/2" drill, enlarged with a 31/32" drill and finished with a 1" reamer. Next, the 3/4" end mill was used to create a flat spot for the mounting bolt's head. Still using the 3/4" end mill, a belt clearance slot was milled into face 3. Finally, the bearing hole was started with the same end mill. The bearing hole was then enlarged with a 27/32" drill and reamed to the required size of 7/8".

In order to support the second 1/4" shaft, a shaft clip had to be added the outer edge of face 2 for the **right** corner block and to face 1 for the **left** corner block. Move the block with the appropriate face up and drill the two holes need to attach the shaft clip. Finally reorient the block with face 3 up and the bottom face against the reference. Attach the shaft clip before drilling. Using a 1/4" drill, drill the hole for the 1/4" shaft.

## Machining - Slider

The slider as shown in Figure 2.7, on page 17, starts as a solid block of aluminum that has been fly cut on all six sides to a finished size of 4.5" by 3" by 3". Table A.3 is a list of all of the tool used to make the slider. With the block in the vise, 4.5" by 3" face up, the four 0.2010 inch holes were drilled to a depth of 3/4". These holes were taped for [1/4]-20 UNC bolts and are used to hold weight plates to the top of the slider. After flipping the block over, a 1/2" wide by 1/8" deep belt slot was cut with a 1/2" end mill. The 4 holes needed for the belt clip were drilled to a depth of 1/2" with a 0.1065" drill bit and tapped to accept [#6]-32 UNC screws.

**Table A-3:** Machine Tools used for making the slider

| Tool | Spindle Speed | Feed Rate | Max Cutting Depth |
|---|---|---|---|
| 0.2010" drill | 1900 rpm | 2.0 inch/min | n/a |
| 0.1065" drill | 3587 rpm | 3.5 inch/min | n/a |
| 1/2" end mill | 760 rpm | 0.8 inch/min | 0.25" |
| 1/2" drill | 760 rpm | 0.8 inch/min | n/a |
| 7/8" drill | 630 rpm | 0.6 inch/min | n/a |
| 1 1/8" drill | 500 rpm | 0.5 inch/min | n/a |
| 1 3/8" drill | 370 rpm | 0.4 inch/min | n/a |
| 1 1/2" drill | 250 rpm | 0.3 inch/min | n/a |
| 5/16" drill | 1200 rpm | 1.2 inch/min | n/a |

To save weight while still keeping a larger payload surface the slider is 'T' shaped. In order to save unnecessary wear on the drills, the band saw was used to remove a two inch by 3/4 inch chunk from the bottom one end of the slider. The slider was placed back in the vice with the cut end up. A 1/2" end mill was used to clean up

the band saw cut. The finished size of the cut was 2.2 inches up from the bottom and 0.8125 inches in from the end. Next the one inch bearing hole was drilled to a full diameter depth of 2.9 inches. Because the hole had a large diameter, the hole had to be drilled with a series of drill bits starting with a 1/2" drill and stepping though the 7/8", 1 1/8", 1 3/8" and finishing with the 1 1/2" drill. By doing this, the area of material removed is approximately a constant of 0.4 $in^2$. A 5/16" drill was used to make the clearance hole for the 1/4" shaft. Again the band saw was used to cut the bulk of the material off the other end. This end was also milled down to the same finished size as the other end. The final step in making the slider was to ream the 1" bearing hole to a finished size of 1.563 inches and to ream out the 1/4" bearing holes to 0.5 inches. Because of the need for accuracy in the bearing holes, the final reaming operation was done in the machine shop.

## Motor Mounting Plate

The only difference between the motor mounting plate used on the SEDP robot and the plate needed for this project is that the motor's mounting holes are in a different location. Size 34 stepping motors, which are much larger than the DC motor used in this project, were used with the original plate. Figure 2.8 on page 19 shows the new spacing required for the DC motor. Table A.4 contains a list of the tools needed to make a motor plate.

To start with an over sized piece of 1/4" aluminum stock was held down to a scrap of plywood on the mill's table. First, the 4 motor mounting holes were drilled with a #28 drill. Next a #P drill was used to make the shaft hole and a pilot hole for both the

**Table A-4:** Machine Tools used for making the motor plate

| Tool | Spindle Speed | Feed Rate | Max Cutting Depth |
|---|---|---|---|
| #28 drill (0.1405") | 2700 rpm | 2.6 inch/min | n/a |
| #P drill (21/64") | 1200 rpm | 1.2 inch/min | n/a |
| 1/4" end mill | 1450 rpm | 1.5 inch/min | 0.125" |

mounting hole and mounting slot. The 1/4" end mill was used to mill out a circular indention for the DC motor and to mill out the mounting slot. Wood screws with washers were used to hold the aluminum plate to the plywood. The same 1/4" end mill was then used again to cut the complex outline of the motor mounting plate.

## MOSFET's Heat-Sink

Because of the large amount of power dissipated by the MOSFET's when the motor is being driven, a heat sink was need for each transistor. In order to save space, one large heat sink was chosen over having individual heat sinks for each transistor. Figure A.2 shows a machine drawing for the heat sink. The heat sink was milled out of a solid piece of 1 inch stock. Using the 3/4" end mill, the transistor side was milled, leaving a raised area for the TO-220 cases to be screwed down to. A #43 drill was used to drill the tap holes for the transistor mounting hardware. After flipping the block over, the slow process of cutting the pin fins was executed using the 1/4" end mill. Table A.5 list all of the tools need to make the pin fin heat sink.

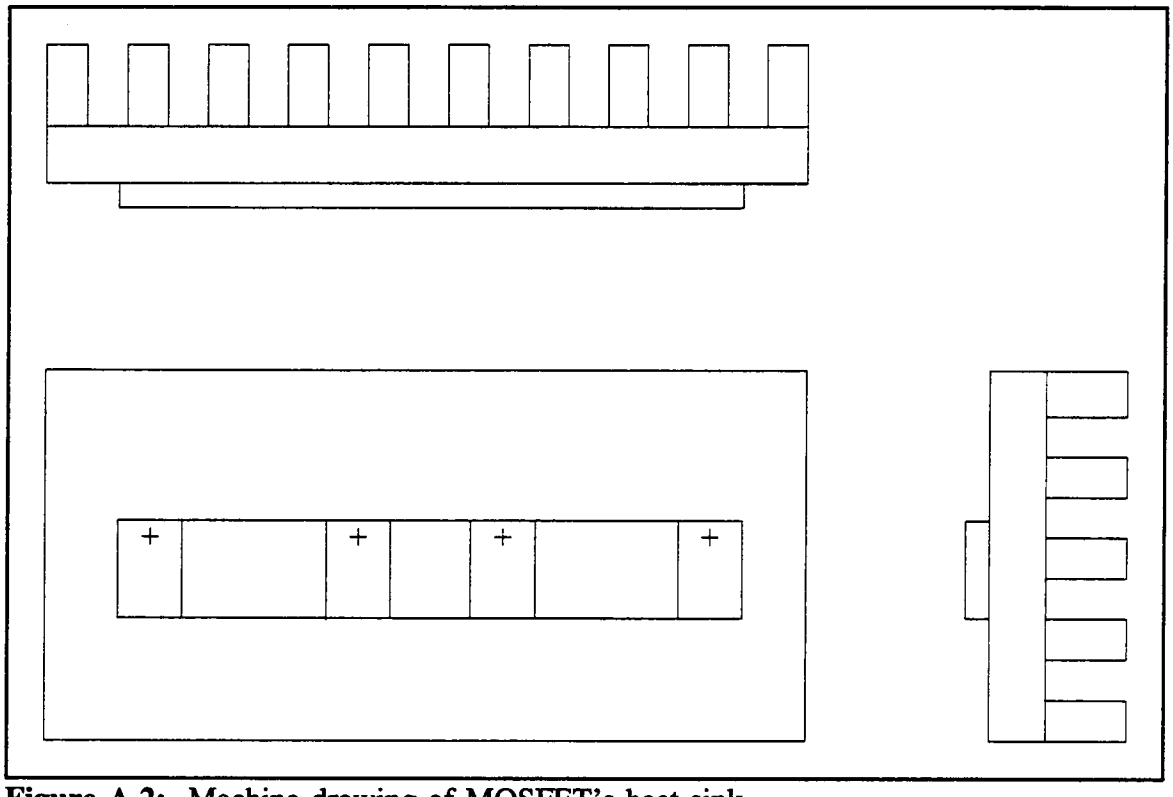


**Figure A.2:** Machine drawing of MOSFET's heat sink.

**Table A-5:** Machine Tools used for making the heat sink.

| Tool | Spindle Speed | Feed Rate | Max Cutting Depth |
|---|---|---|---|
| #43 drill (0.089") | 3600 rpm | 3.7 inch/min | n/a |
| 3/4" end mill | 450 rpm | 0.5 inch/min | 0.375" |
| 1/4" end mill | 1450 rpm | 1.5 inch/min | 0.125" |

## Circuit Cards

There were four major hand made circuit cards manufactured for this project. They are the H-Drive card, the $|A-B|$ card, the user interface card, and the quadrature decoder card. The layout for each of these cards is shown in Figures A.3 through A.6 respectively. In each layout, the dotted lines represent the under side circuit traces and the solid lines represent the traces on top side.

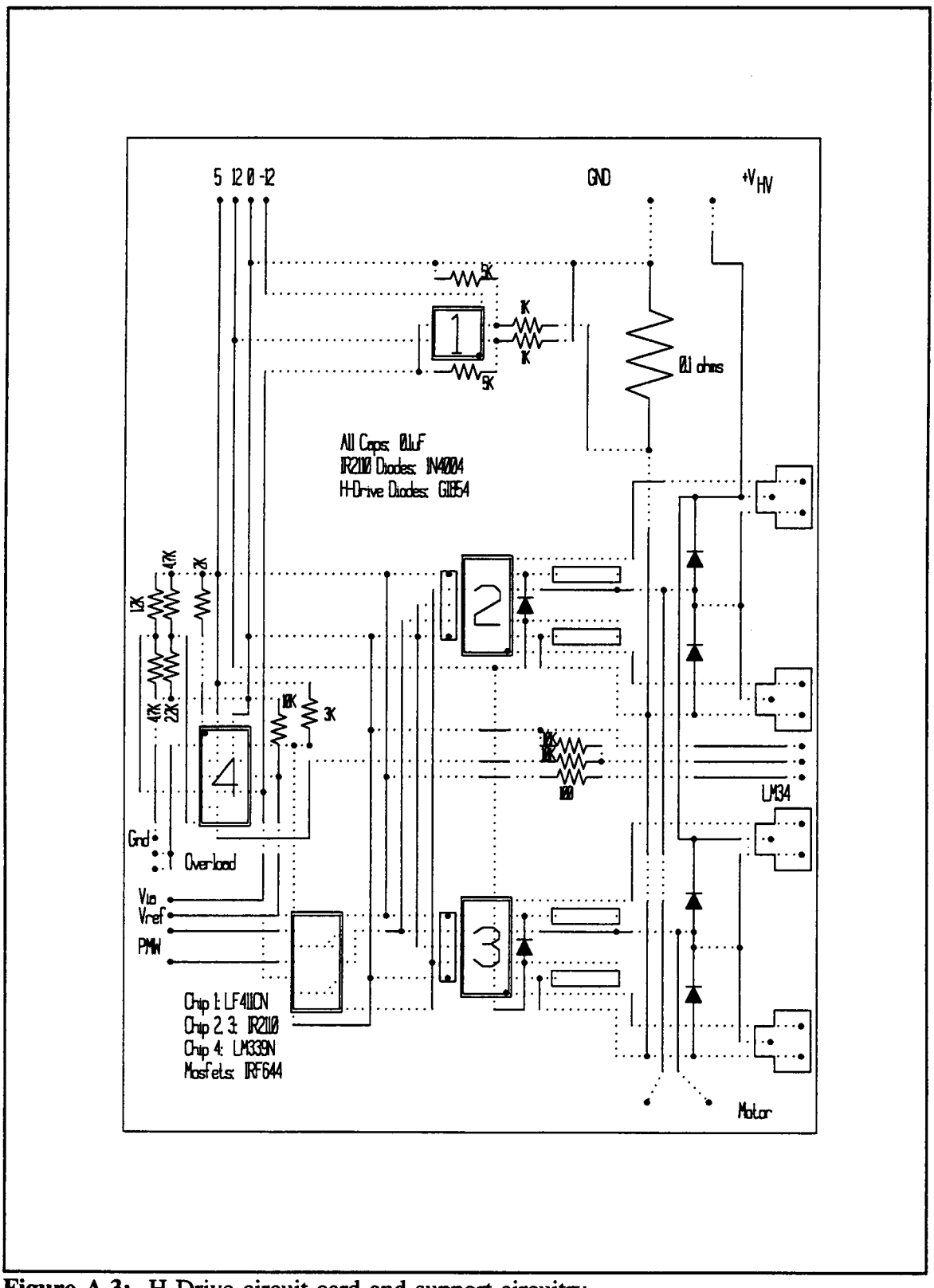



**Figure A.3:** H-Drive circuit card and support circuitry.

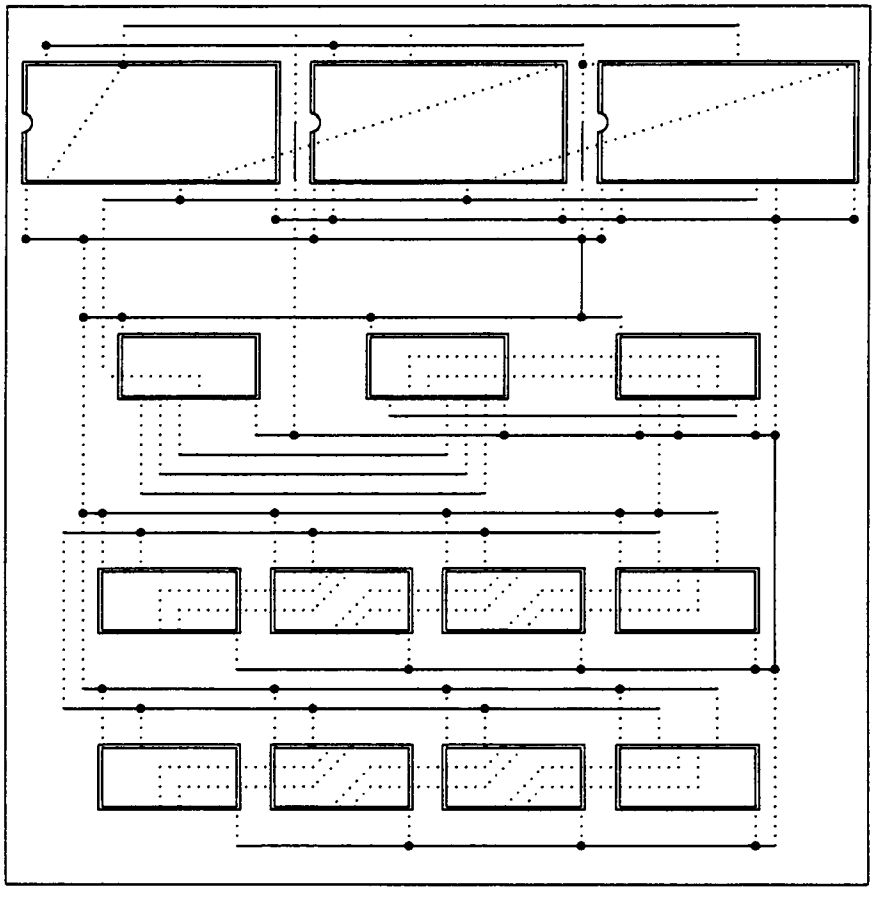


**Figure A.4:** Math card used to get $|A-B|$ and dual 16 bit counters for **A** and **B**.

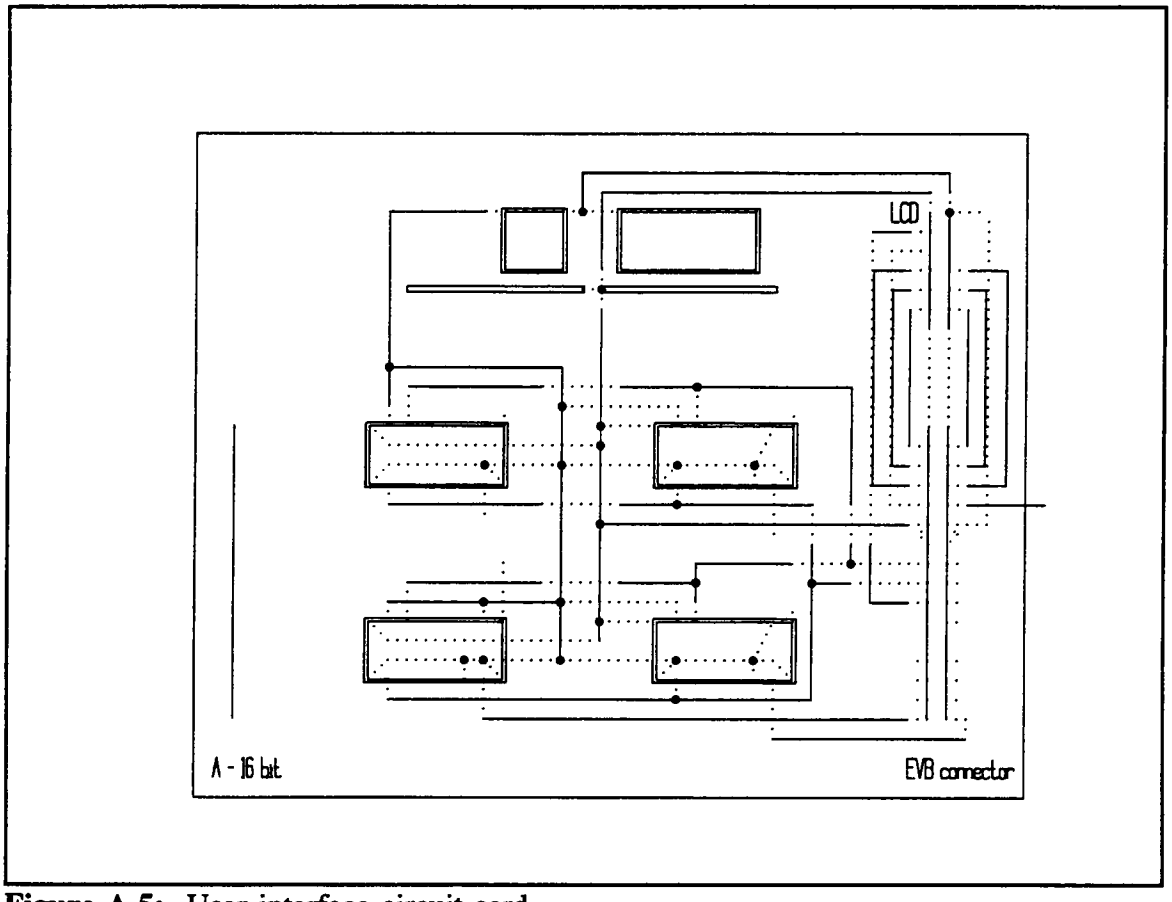



**Figure A.5:** User interface circuit card.

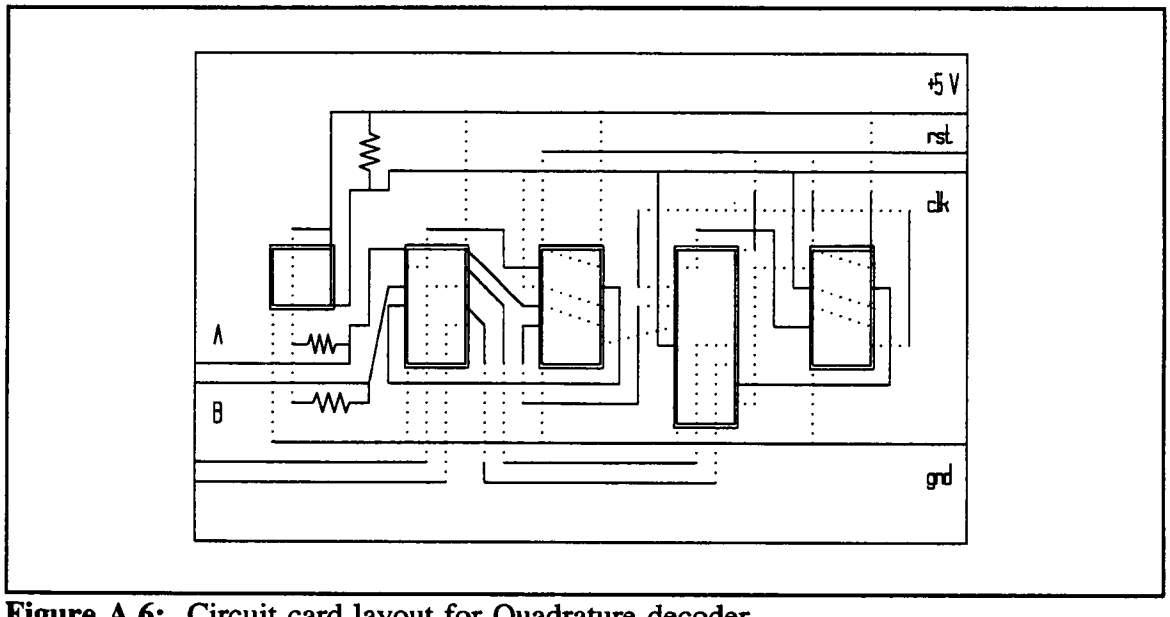



**Figure A.6:** Circuit card layout for Quadrature decoder.

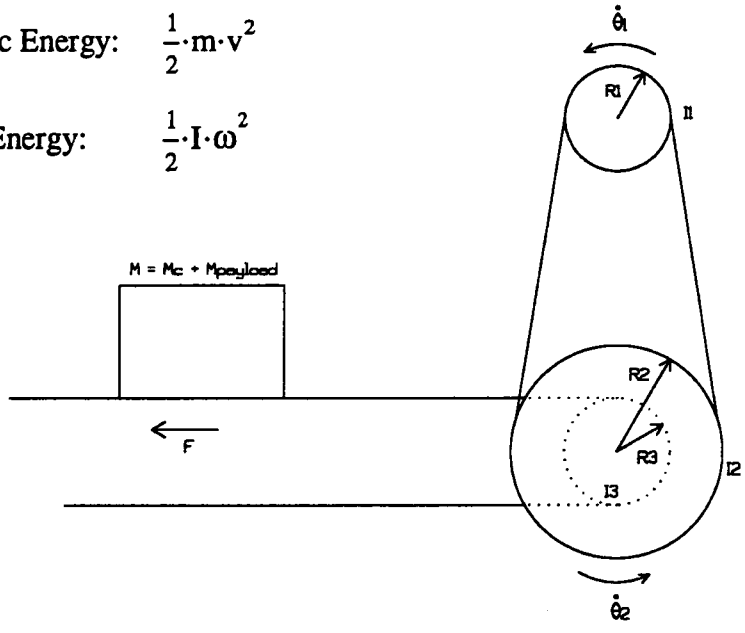
# Appendix B

This appendix includes the calculations and derivations for the creation of a controls model and simulation. The calculations and plots shown below were generated with Mathcad 5.0 plus.

# Calculation of Equivalent System Inertia

Translational Kinetic Energy: $\frac{1}{2}\cdot m\cdot v^2$

Rotational Kinetic Energy: $\frac{1}{2}\cdot I\cdot \omega^2$




Inertia for pulley 1 and 3:

destiny of steel: $\rho := 0.238\cdot\frac{lb}{in^3}$ pulley radius: $r_1 := 0.4775\cdot in$

pulley thickness: $t_1 := 0.5\cdot in$ gravity: $g := 386.4\cdot\frac{in}{sec^2}$

pulley weight: $w_1 := \pi\cdot r_1^2\cdot t_1\cdot\rho$ $w_1 = 1.364\cdot oz$

$I_1 := 0.5\cdot\frac{w_1}{g}\cdot r_1^2$ $I_1 = 4.024\cdot 10^{-4}\cdot in\cdot oz\cdot sec^2$

$I_3 := I_1$ $r_3 := r_1$

Inertia for pulley 2:

pulley radius: $r_2 := 0.955 \cdot in$ pulley thickness: $t_2 := 0.5 \cdot in$

pulley weight: $w_2 := \pi \cdot r_2^2 \cdot t_1 \cdot \rho$ $w_2 = 5.455 \cdot oz$

$$I_2 := 0.5 \cdot \frac{w_2}{g} \cdot r_2^2 \qquad I_2 = 0.006 \cdot in \cdot oz \cdot sec^2$$

Inertia contribution from the mass of the slider and payload:

Weight of the slider: $w_{sl} := 3 \cdot lb$

Weight of the payload: $w_{pl} := 5 \cdot lb$

$$Iw := \frac{w_{sl} + w_{pl}}{g} \cdot r_3^2 \qquad Iw = 0.076 \cdot in \cdot oz \cdot sec^2$$

Inertia of the motor: $Im := 0.0011066 \cdot in \cdot oz \cdot sec^2$

Belt Ratio: $BR := \frac{1}{2}$

Total system inerta as seen by the motor:

$$Is := Im + I_1 + BR^2 \cdot (I_2 + I_3 + Iw) \qquad Is = 0.0221 \cdot in \cdot oz \cdot sec^2$$

## Load Torque

Forces applied to the slider can be reflected back to the motor in the form of a disturbance in the available torque.

Disturbance force: $f := 0.25 \cdot lb$

Resulting Load Torque: $TL := f \cdot \frac{r_1 \cdot r_3}{r_2}$ $TL = 0.955 \cdot in \cdot oz$

# Mathmatical System Model Derivation

Two assumptions were made to simplify the design process. The disturbance tourque is assumed to be zero. Because the motor is only rated for 12 volts, it's back emf can be neglected when the motor is being powered with a 36 to 180 volt switching amplifier.

Using the equation of $Gcl(s) := \frac{G(s)}{1 + H(s)\cdot G(s)}$ the transfere function of the closed loop system can be determined

Transfere function of the motor: $G(s) := \frac{6160}{s\cdot(s + 8.135)}$

The amplifier can be idealized to a constant: $K := 0.5\cdot\frac{volt}{step}$

Closed loop transfere function: $G_{cl}(s) := \frac{3080}{s^2 + 8.135\cdot s + 3080}$

From the ACSL simulation of the actualy system including the switching amplifier the time to the first peak was determined to be 0.08 seconds with 67% overshoot.

Time to first peak: $t_p := 0.08\cdot sec$

Percent Overshoot: $OS := 66.7$

Closed loop bandwidth: $\omega_{BW} := \frac{4.4}{t_p}$ $\omega_{BW} = 55\cdot\frac{rad}{sec}$

Damping factor: $\zeta := \frac{-\ln\left(\frac{OS}{100}\right)}{\sqrt{\ln\left(\frac{OS}{100}\right)^2 + \pi^2}}$ $\zeta = 0.128$

Choosing a value to increase the phase margin by: $\phi_m := 60 \cdot deg$

$$a := \frac{\sin(\phi_m) + 1}{1 - \sin(\phi_m)} \qquad a = 13.928$$

Maximum compensator gain: $20 \cdot \log(a) = 22.878$ dB

Locate frequency on original bode plot were gain is equal to half of the negative value of maximum of the compensator (-11.45 dB).

$$\omega_m := 190 \qquad T := \frac{1}{\sqrt{a} \cdot \omega_m} \qquad T = 0.00141$$

Transfere function of the compensator:

$$Gc(s) := \frac{s + \frac{1}{a \cdot T}}{s + \frac{1}{T}} = \frac{s + 50.78}{s + 711}$$

Because the compensator is implemented in a computer program, the compensator needs to be writen in the form of a difference equation.

Using a z-transform the compensator can be written as:

$$Gc(z) := \frac{z - 0.955}{z - 0.37} = \frac{Y(nT)}{X(nT)}$$

The compensator written as a difference equation:

$$Y(nT) := 0.37 \cdot Y((n - 1) \cdot T) + X(nT) - X((n - 1) \cdot T)$$

Using mathcad bode plots of the system and compensator are generated below. The first few lines below are common to each case and will not be repeated.

gain: $db(G,\omega) := 20\cdot\log(|G(j\cdot\omega)|)$

phase shift: $ps(G,\omega) := \frac{180}{\pi}\cdot\arg(G(j\cdot\omega)) - 360\cdot(\text{if}(\arg(G(j\cdot\omega))\geq 0,1,0))$

min freq: $S := .1$ max freq: $H := 1000$ number of points: $N := 200$

$i := 0..N$ $r := \log\left(\frac{H}{S}\right)\cdot\frac{1}{N}$ $\omega_i := S\cdot 10^{i\cdot r}$

## 2nd order appoximation of the Open Loop System

Open-loop transfer function: $K := 0.5$

$$G(s) := \frac{3080\cdot K}{s\cdot(s+8.135)}$$

Enter a guess for the gain crossover frequency: $\omega_c := 1$

Solve for the gain crossover frequency: $\omega_c := root(db(G,\omega_c),\omega_c)$

$\omega_c = 38.823$

Calculate the phase margin pm: $pm := ps(G,\omega_c) + 180$

$pm = 11.835$

Estimate frequency at 0 db: $\omega_0 := 10000$

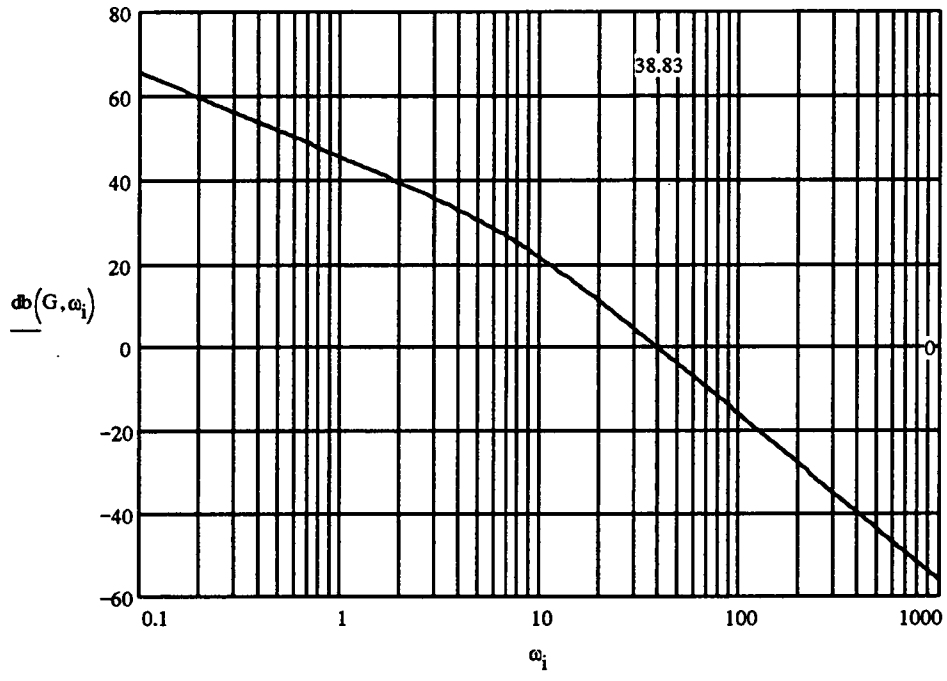
Solve for ω at 0 db using this estimate: $\omega_0 := root(ps(G,\omega_0) + 180,\omega_0)$

$\omega_0 = 5.503\cdot 10^5$

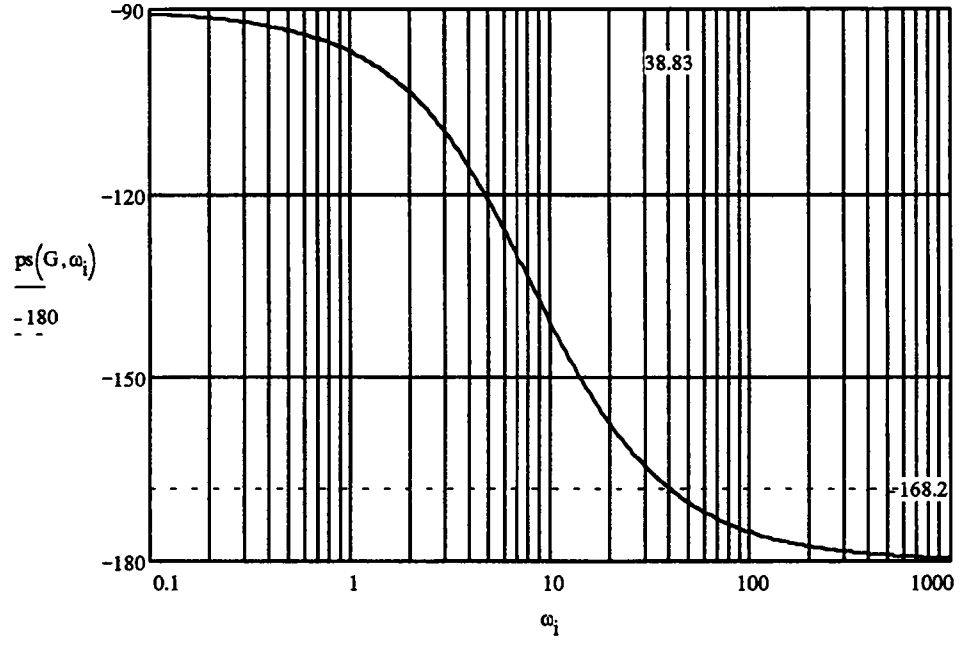
Calculate the gain margin gm: $gm := -db(G,\omega_0)$

$gm = 165.875$

Plot the open-loop gain in db as a function of frequency:




Plot the phase shift as a function of frequency:



# Closed loop bode plot

Closed loop transfer function: $K := 0.5$

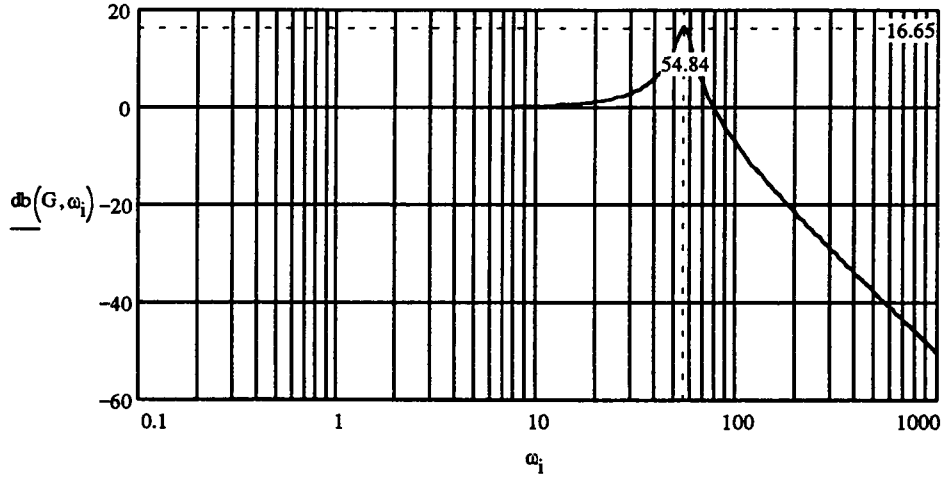
$$G(s) := \frac{6080 \cdot K}{s^2 + 8.135 \cdot s + 6080 \cdot K}$$

Find peak magnitude of the CLS bode plot and it's frequency:

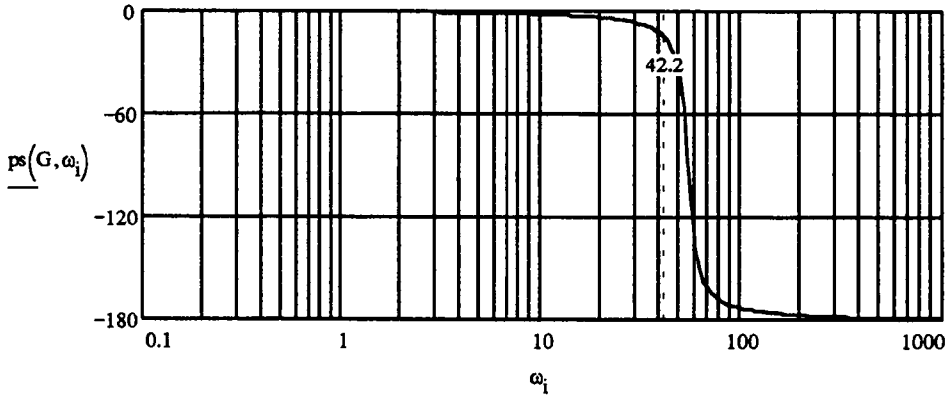
$rf := 54$ $\quad rf := \text{root}\left(\frac{d}{drf}\left|db(G,rf)\right|, rf\right)$ $\quad rf = 54.835$

$Mp := db(G,rf)$ $\quad Mp = 16.645$

Gain plot of the closed loop system in dB:




Phase plot of the closed loop system:

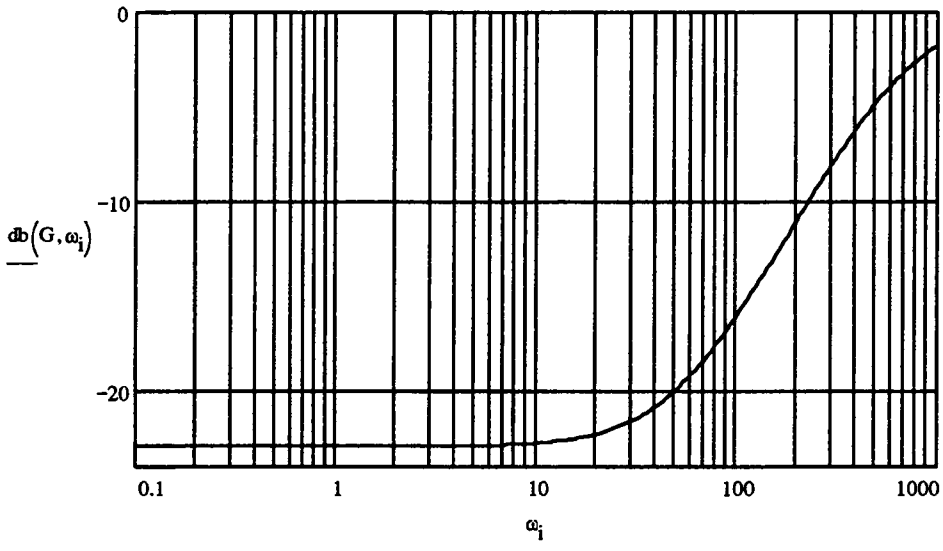


# Bode Plot of the Compensator

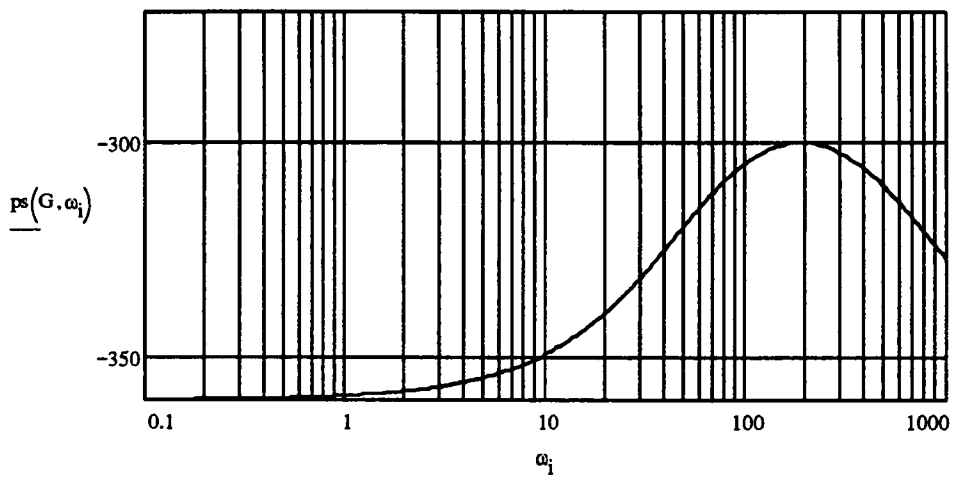
Compensator transfer function: $a := 14$ $T := 0.001407$

$$G(s) := \frac{s + 50.78}{s + 711.0}$$

Gain plot of the compensator in dB:




Phase plot of the compensator:



# Open loop bode plot of the system and compensator

Open-loop transfer function: $K := 0.5$

$$G(s) := \frac{6080 \cdot K \cdot (s + 50.78)}{s \cdot (s + 8.135) \cdot (s + 711)}$$

Enter a guess for the gain crossover frequency: $\omega_c := 1$

Solve for the gain crossover frequency: $\omega_c := \text{root}(\text{db}(G, \omega_c), \omega_c)$

$\omega_c = 13.944$

Calculate the phase margin pm: $pm := \text{ps}(G, \omega_c) + 180$

$pm = 44.491$

Estimate frequency at 0 db: $\omega_0 := 10000$

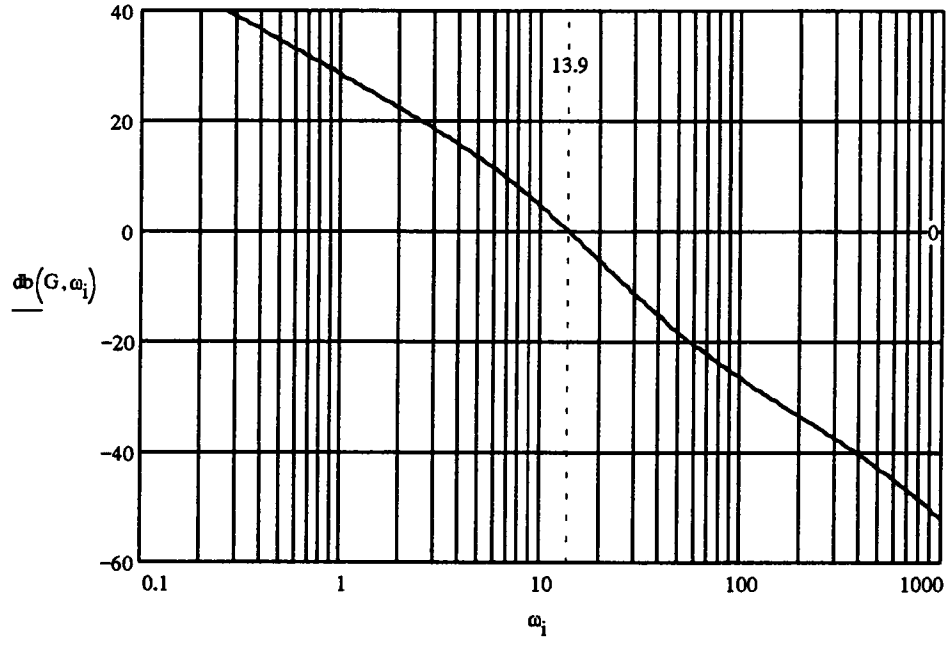
Solve for ω at 0 db using this estimate: $\omega_0 := \text{root}(\text{ps}(G, \omega_0) + 180, \omega_0)$

$\omega_0 = 4.193 \cdot 10^7$

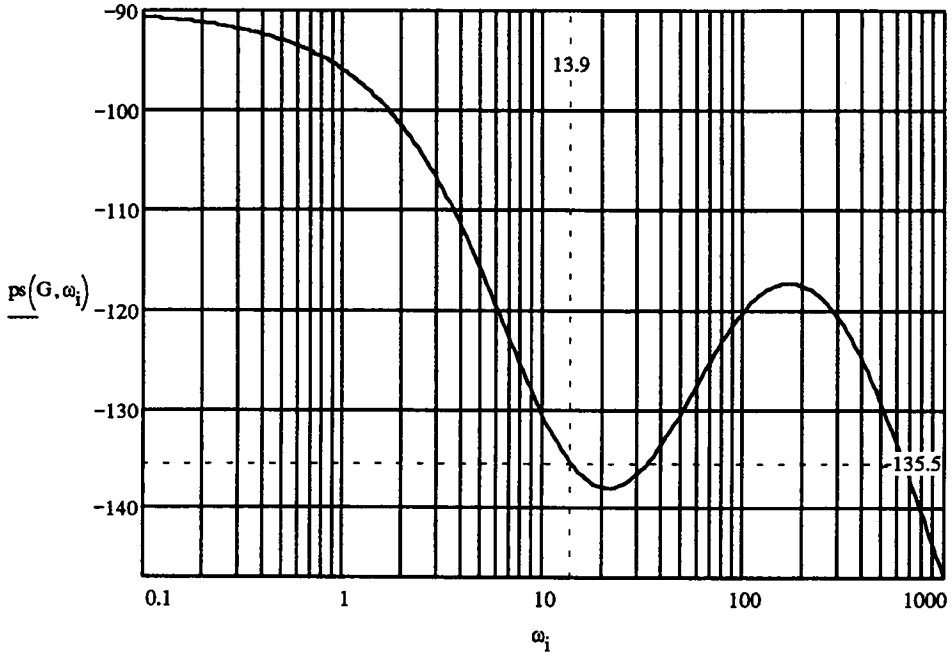
Calculate the gain margin gm: $gm := -\text{db}(G, \omega_0)$

$gm = 235.245$

Gain plot of the system and compensator in dB:




Phase plot of system and compensator:



# Closed loop bode plot of compensator and system

Closed loop transfer function: $K := 0.5$

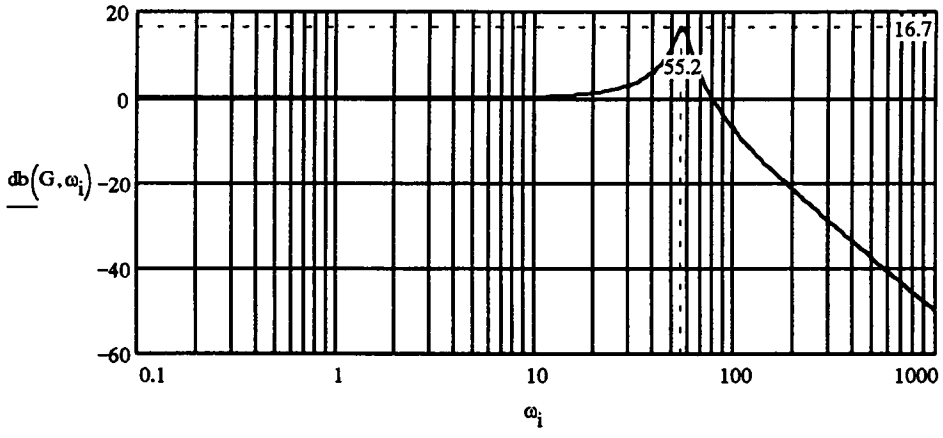
$$G(s) := \frac{6160 \cdot K \cdot (s + 50.78)}{s \cdot (s + 8.135) \cdot (s + 50.78) + 6160 \cdot K \cdot (s + 50.78)}$$

Find peak magnitude of the CLS bode plot and it's frequency:

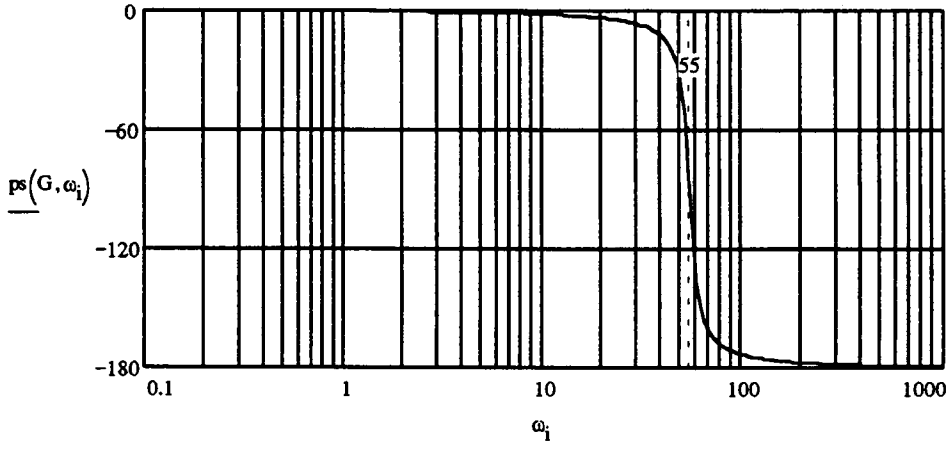
$rf := 54$ $\quad rf := \text{root}\left(\frac{d}{drf}\left|db(G, rf)\right|, rf\right)$ $\quad rf = 55.199$

$Mp := db(G, rf)$ $\quad Mp = 16.702$

Gain plot of the closed loop system in dB:




Phase plot of the closed loop system:



# Shafting sizing calculations

One of the recomendations from Chapter 5 was to resize the shafting used in the test stand. Below are deflection calculations for various size shafts.

Test load: $F := 20 \cdot lbf$

Diameter of the shafting: $D := 0.375 \cdot in$

Total Length of shafting: $L := 36 \cdot in$

Modulus of elasticity for steel: $E := 30 \cdot 10^6 \cdot psi$

Second moment of the area: $I := \frac{\pi \cdot D^4}{64}$

For a concentrated load on a pin-pin beam deflection is: $Def := \frac{F \cdot L^3}{192 \cdot E \cdot I}$

$Def = 0.167 \cdot in$

# Appendix C

## Software listing for the 'PC Interface'

```
*       File:           PCINTR.asm
*       Date:           6/2/94
*       Version:        2.0  Internal Eprom Version
*       Programmer:     Mark Holdaway
*       Contains:       Interface routines to connect a PC parallel port
*                       to the drive system
*
*       Hardware Connections from PC:
*                               STEP            PORTC   bit 0   (9)
*                               DIR             PORTC   bit 1   (10)
*                               RESET   PORTC   bit 2   (11)
*                               Axis Ready      PORTB   bit 4   (38)
*
*       Hardware Connections from Drive system:
*                               UP              PORTB   bit 0   (42)
*                               DOWN    PORTB   bit 1   (41)
*                               RST_CNTR        PORTB   bit 2   (40)
*                               Axis Zero Ind.  PORTC   bit 3   (12)
*                               RST_QUAD        PORTB   bit 3   (39)
*
**Equates - Registers will be addressed with Ind,X mode
*
REGBAS  EQU     $1000       Starting address for register block
PORTB   EQU     $04         Output Port B
PORTC   EQU     $03         Input Port C

        ORG     $C000       Program starts in EVB RAM at $C000
***
* Program begins here
***
START   LDS     #$0047              Load stack pointer for user stack on EVB
        LDX     #REGBAS             Pointer to register block
        LDAA    #$07                Reset all the drive
        STAA    PORTB,X             system outputs
        LDAA    #$0B                Release counter and quad decoder
        STAA    PORTB,X             resets while keeping UP & DOWN high

AGAIN   BRSET   PORTC,X $04 HME     if RESET is high go to HOME routine
        JSR     STEP                else go to step routine
        BRA     AGAIN               continue looping
HME     JSR     HOME                jump to HOME routine
        BRA     AGAIN               continue looping
END     JMP     $E0B2               End program and return control to BUFFALO
```

```
***
* HOME - Homes axis when reset signal is received
***

HOME    LDAA   #$07              Reset counters and quad decoder
        STAA   PORTB,X
        LDAA   #$0B              Ready counters and quad decoder
        STAA   PORTB,X

        LDY    #REGBAS           load register pointer into Y
        LDX    #$32              50 steps forward
UP50    BCLR   PORTB,Y $01       set CNT UP high
        JSR    DLY125            delay 1.25ms
        BSET   PORTB,Y $01       reset CNT UP low
        JSR    DLY125            delay 1.25ms
        DEX                      dec X
        BNE    UP50              if 50 steps aren't up continue
        LDX    #REGBAS           reload register pointer

AZI     BRSET  PORTC,X $08 AR    if AZI already detected continue on
        BCLR   PORTB,X $02       set CNT DOWN high
        JSR    DLY125            delay 1.25ms
        BSET   PORTB,X $02       reset CNT DOWN low
        JSR    DLY125            delay 1.25ms
        BRA    AZI               continue looping until AZI detected

AR      BSET   PORTB,X $10       set axis ready high
RST     BRSET  PORTC,X $04 RST   loop until reset released
        BCLR   PORTB,X $10       set axis ready low
        JSR    DLY125            extra delay to prevent false re-resetting

        LDAA   #$07              Reset counters and quad decoder
        STAA   PORTB,X
        LDAA   #$0B              Ready counters and quad decoder
        STAA   PORTB,X
        RTS                      end of subroutine
```

```
***
* STEP - Convert the Step and Dir inputs to UP/DOWN outputs
***

STEP    BRCLR PORTC,X $01 SKIP  skip routine if STEP is low
        BRSET PORTC,X $02 UP    if DIR is high then count UP
        BCLR  PORTB,X $02       else count DOWN
        BRA   WAIT              jump to wait until STEP goes low
UP      BCLR  PORTB,X $01       count UP
WAIT    BRSET PORTC,X $01WAIT   wait for STEP to go low
        BSET  PORTB,X $03       set UP/DOWN low after STEP goes low
SKIP    RTS                     end of subroutine


***
* DLY125 - Subroutine to delay 1.25mS (for E=2MHz)
***

DLY125  PSHX                    Save the value of X
        LDX   #$F1A0            416 * 6 * 0.5uS = 1.25mS delay
D125LP  DEX                     [3] [# of clock cycles per instruction]
        BNE   D125LP            [3] continue for 3334 times
        PULX                    Recover X value
        RTS                     end of subroutine
```

## Software listing for the '|A-B| EPROM'

```
/*********************
 * EPROM11.C*
 *********************/

#include "stdio.h"
#include "conio.h"
#include "math.h"

void main(void)
   {
   FILE *fp;
   int A, B, AGTB, CIN, Q, COUT, OUT;
   int chkadd=0, chksum=0;
   int a, b, j=0, addr=0, addrh, addrl;
   clrscr();
   fp = fopen("part1.hex", "w");
   fprintf(fp, ":10000000");
   for(A=0; A<=15; A++)
      {
      for(B=0; B<=15; B++)
         {
         for(AGTB=0; AGTB<=1; AGTB++)
            {
            for(CIN=0; CIN<=1; CIN++)
               {
               a = A - AGTB*CIN;
               b = B - (1-AGTB)*CIN;
               COUT=0;
               if((AGTB==1 && (a-b)<0)||(AGTB==0 && (b-a)<0)) COUT=1;
               a = a + 16 * AGTB * COUT;
               b = b + 16 * (1-AGTB) * COUT;
               Q=abs(a-b);
               OUT = Q*16+COUT*8;
               if(j<=15)
                  {
                  chkadd = chkadd + OUT;
                  fprintf(fp, "%2X", OUT);
                  j++;
                  }
               else
                  {
                  addrh = (addr*16 & 0xFF00) / 256;
                  addrl = addr*16 & 0x00FF;
                  chksum = -(chkadd + addrh + addrl + 16) & 0xFF;
                  addr++;
                  fprintf(fp, "%2X\n:10%3X000%2X", chksum, addr, OUT);
                  chkadd = OUT;
                  j=1;
                  }
               }
            }
         }
      addrh = (addr*16 & 0xFF00) / 256;
      addrl = addr*16 & 0x00FF;
      chksum = -(chkadd + addrh + addrl + 16) & 0xFF;
      fprintf(fp, "%2X", chksum);
   }  /* End of main */
```

## Software listing for the 'User Interface'

```
*       File:           Userintr.asm
*       Date:           6/1/94
*       Version:        2.00 EPROM VERSION ADDR 0x6000
*       Programmer:     Mark Holdaway
*       Contains:       Interface routines to a 2-line X 20
*                       character Phillips LCD display LTN221
*
*       Hardware Connections from display to 68HC11:
*               Register Select     PORTA  bit 5     (29)
*               Enable (Data Latch) STRB             (6)
*               I/O Data bits 0-7   PORTB  bits 0-7  (42-35)
*
*       Hardware Connections from 4-1 Mux to 68HC11:
*               Register Select     PORTA  bits 3-4  (31-30)
*               I/O Data bits 0-7   PORTC  bits 0-7  (9-16)
*
*       Hardware Connections from User Controls to 68HC11:
*               Load New Current    PORTA  bit 7     (27)
*               Display Position    PORTA  bit 0     (34)
*               Display Motor Cur   PORTA  bit 1     (33)
*               Display Setpoint Cur PORTA bit 2     (32)
*
*       Hardware Connections from PWM 68HC11 to User I/O 68HC11:
*               TxD - SCI send line PORTD  bit 0     (20)
*               Motor Current (Via) PORTE  bit 2     (47)
*
**Equates - Registers will be addressed with Ind,X mode
*
REGBAS  EQU     $1000   Starting address for register block
PORTA   EQU     $00     Data register Port A
PORTB   EQU     $04     Output Port B
PORTC   EQU     $03     Input/Output Port C
ADCTL   EQU     $30     A/D Control Register
ADR3    EQU     $33     A/D Result Register 3
PIOC    EQU     $02     Parallel I/O Control Register
RS      EQU     $20     Port A bit five connected to RS on display
MUX1    EQU     $08     Mask for Port A bit 3
MUX2    EQU     $10     Mask for Port A bit 4
MUX3    EQU     $18     Mask for Port A bits 3 and 4
BAUD    EQU     $2B     SCI BAUD rate control register
SCCR1   EQU     $2C     SCI control register 1
SCCR2   EQU     $2D     SCI control register 2
SCSR    EQU     $2E     SCI status register
SCDR    EQU     $2F     SCI data register
```

```
***
* Program begins here
***

        ORG     $6000               Prog starts in EVB extra socket at $6000
START   LDS     #$0047              Load stack pointer for user stack on EVB
        LDX     #REGBAS             Pointer to register block
        JSR     INIT                Initialize LCD display
        JSR     INIT2               Initialize SCI port
MENU    BRSET   PORTA,X $07 OP2     Jump to Position subroutine
        BRSET   PORTA,X $05 OP3     Jump to Motor Current subroutine
        BRSET   PORTA,X $03 OP4     Jump to Current Set point subroutine
        BRSET   PORTA,X $06 OP1     Jump to Belt Ratio subroutine
OP1     JSR     RATIO               Belt Ratio
        BRA     MENU                recheck input selector
OP2     JSR     POS                 Position
        BRA     MENU                recheck input selector
OP3     JSR     MCUR                Motor Current
        BRA     MENU                recheck input selector
OP4     JSR     MCSET               Current Setpoint
        BRA     MENU                recheck input selector
END     JMP     $E0B2               End program and return control to BUFFALO


***
* INIT: initializes the LCD display after power-up
***

INIT    LDAA    #$30                set A/D to read PORT E 0-3
        STAA    ADCTL,X             store in A/D control reg
        LDAA    #$02                set strobe B to active low
        STAA    PIOC,X              store in parallel I/O control reg
        BCLR    PORTA,X RS          set RS pin low

        JSR     DLY10               first 10mS of min 15mS power-up delay
        JSR     DLY10               total of 20mS of power-up delay
        LDAA    #$30                Function Set to 8-bit interface
        STAA    PORTB,X             write value to Port B
        JSR     DLY10               Delay 4.1mS or more
        LDAA    #$30                Function Set to 8-bit interface
        STAA    PORTB,X             write value to Port B
        JSR     DLY10               Delay 100uS or more
        LDAA    #$30                Function Set to 8-bit interface
        STAA    PORTB,X             write value to Port B
        JSR     DLY40               Delay 40uS after a display command
```

```
        LDAA    #$38            Final Function Set
        STAA    PORTB,X         write value to Port B
        JSR     DLY40           Delay 40uS after a display command

        LDAA    #$08            Display off
        STAA    PORTB,X         write value to Port B
        JSR     DLY40           Delay 40uS after a display command
        LDAA    #$01            Display clear
        STAA    PORTB,X         write value to Port B
        JSR     DLY10           Delay > 1.64mS after a display clear
        LDAA    #$06            Set to write across the display
        STAA    PORTB,X         write value to Port B
        JSR     DLY40           Delay 40uS after a display command
        LDAA    #$0C            Turn display and with no cursor
        STAA    PORTB,X         write value to Port B
        JSR     DLY40           Delay 40uS after a display command
        RTS                     End of subroutine

***
* INIT2: initializes the SCI port
***

INIT2   LDAA    #$30            data to set highest baud to 9600
        STAA    BAUD,X          set baud rate to 9600 for E = 2.0 MHz
        LDAA    #$00            data to set mode to:
        STAA    SCCR1,X         1 start, 8 data, 1 stop bit
        LDAA    #$0C            data to enable the transmitter
        STAA    SCCR2,X         and receiver
        LDAA    #$00            Zero drive current on reset
SCI     LDAB    SCSR,X          Read SCSR register
        BITB    #$80            Isolate the TDRE bit
        BEQ     SCI             if Transmitter register not empty loop
        STAA    SCDR,X          store current to SCI data register - TxD
        RTS
```

```
***
* PRINT: Writes the string pointed to by index reg y to the LCD.
*        The string should be terminated by a 0.
***

PRINT   LDAA  #$01        Display clear
        STAA  PORTB,X     write value to Port B
        JSR   DLY10       Delay of 1.64mS or more after a display clear
        LDAA  #$02        Home display
        STAA  PORTB,X     write value to Port B
        JSR   DLY10       Delay of 1.64ms or more after home display
        BSET  PORTA,X RS  set RS bit on display to data mode (high)
NEXT    LDAA  0,Y         load character in accum. A
        BEQ   DONE        if character is nll the jump to DONE
        STAA  PORTB,X     print character
        INY               move pointer to the next character
        JSR   DLY40       delay 40uS after a display command
        JMP   NEXT        continue looping until done
DONE    BCLR  PORTA,X RS  set RS bit on display to control mode (low)
        RTS               end of subroutine

***
* PRINT2: Writes the string pointed to by index reg y to the LCD without changing the
* position on the LCD. The string should be terminated by a 0.
***

PRINT2  LDX   #REGBAS     Pointer to register block
NEXTP2  LDAA  0,Y         load character in accum. A
        BEQ   DONEP2      if character is nll the jump to DONE
        STAA  PORTB,X     print character
        INY               move pointer to the next character
        JSR   DLY40       delay 40uS after a display command
        JMP   NEXTP2      continue looping until done
DONEP2  BCLR  PORTA,X RS  set RS bit on display to control mode (low)
        RTS               end of subroutine

***
* LINE2: Moves the display address pointer to the beginning of the second line.
***

LINE2   BCLR  PORTA,X RS  set RS bit on display to control mode (low)
        LDAA  #$C0        Move to beginning of 2nd line
        STAA  PORTB,X     write value to Port B
        JSR   DLY40       Delay of 40uS after an address set
        BSET  PORTA,X RS  set RS bit on display to data mode (high)
        LDY   #REGBAS     Pointer to register block
        RTS               end of subroutine
```

```
***
* THOU:  Separates and prints the 1000's digit for a number in D
***

THOU    LDX    #1000          1000 used to separate 1000's digit
        IDIV                  Current/1000 -> X; r -> D
        XGDX                  Save r in X and 1000's digit in D
        ADDB   #$30           Convert to ASCII
        STAB   PORTB,Y        Write value to display
        JSR    DLY40          delay 40us
        XGDX                  place r back into D
        RTS                   end of subroutine


***
* HUND:  Separates and prints the 100's digit for a number in D
***

HUND    LDX    #100           100 used to separate 100's digit
        IDIV                  Current/100 -> X; r -> D
        XGDX                  Save r in X and 100's digit in D
        ADDB   #$30           Convert to ASCII
        STAB   PORTB,Y        Write value to display
        JSR    DLY40          delay 40us
        LDAA   #$2E           Hex value for decimal point
        STAA   PORTB,Y        Write value to display
        JSR    DLY40          delay 40us
        XGDX                  place r back into D
        RTS                   end of subroutine


***
* TENZ:  Separates and prints the 10's and 1's digits for a number in D
***

TENZ    LDX    #10            10 used to separate 10's digit
        IDIV                  Value/10 -> X; r -> D
        XGDX                  Save r in X and 10's digit in D
        ADDB   #$30           Convert to ASCII
        STAB   PORTB,Y        Write value to display
        JSR    DLY40          delay 40us
        XGDX                  Value/1 (r) -> D; junk -> X
        ADDB   #$30           Convert to ASCII
        STAB   PORTB,Y        Write value to display
        JSR    DLY40          delay 40us
        RTS                   end of subroutine
```

```
***
* Ratio:  Displays belt ratio in X to 1 format with update every .1 sec
***

RATIO   LDY    #TEXT1            Pointer to message for Ratio
        JSR    PRINT             print message
        BCLR   PORTA,X MUX3      set MUX to read ratio
C7      JSR    LINE2             set up 2nd line of display
        BRSET  PORTC,Y $03 R3    ratio of 3
        BRSET  PORTC,Y $02 R2    ratio of 2
        BRSET  PORTC,Y $01 R1    ratio of 1
        LDY    #RATIO0           pointer to ratio 0 message
        JSR    PRINT2            print message
        BRA    MORE              jump do delay before looping
R1      LDY    #RATIO1           pointer to ratio 1 message
        JSR    PRINT2            print message
        BRA    MORE              jump do delay before looping
R2      LDY    #RATIO2           pointer to ratio 2 message
        JSR    PRINT2            print message
        BRA    MORE              jump do delay before looping
R3      LDY    #RATIO3           pointer to ratio 3 message
        JSR    PRINT2            print message
MORE    JSR    DLY100            Delay 100ms
        BRSET  PORTA,X $07 CA    Needed to make sure program can exit
        BRSET  PORTA,X $06 C8    If Port A bits 0-2 still '110' continue
CA      BCLR   PORTA,X MUX3      reset MUX
        BRA    C9                exit loop
C8      JMP    C7                continue looping
C9      RTS                      end of subroutine


***
* Position:  Displays axis position in inches with update every 0.1 sec
***

POS     LDY    #TEXT2            pointer to position title
        JSR    PRINT             print title
C1      JSR    LINE2             setup 2nd line of display
        BCLR   PORTA,Y MUX3      reset MUX
        LDAA   #$04              Constant 4
        LDAB   PORTC,Y           Read in Belt Ratio
        MUL                      multiply Belt Ratio by 4
        STD    RATX4
        BSET   PORTA,Y MUX2      set MUX to read A-LSB
        LDAA   PORTC,Y           read A-LSB
        LDAB   #$03              Scaling Factor (SF)
        MUL                      multiply SF * A-LSB
```

```
        XGDX                    SF * A-LSB -> X for storage
        BCLR    PORTA,Y MUX3    reset MUX
        BSET    PORTA,Y MUX1    set MUX to read A-MSB
        LDAA    PORTC,YMSB
        LDAB    #$03            SF
        MUL                     multiply SF * A-MSB
        TBA                     transfer Accum B into Accum A
        LDAB    #$00            zero out Accum B
        STX     MATH1           store SF * A-LSB into MATH1
        ADDD    MATH1           add SF * A-MSB and SF * A-LSB
        LDX     RATX4           Belt Ratio * 4
        IDIV                    Position MSB -> X; Pos LSB -> D
        STAB    POSLSB          Save Position LSB
        XGDX                    Move MSB to D
        LDX     #10000          10000 used to separate 10000's digit
        IDIV                    Current/10000 -> X; r -> D
        XGDX                    Save r in X and 10000's digit in D
        ADDB    #$30            Convert to ASCII
        STAB    PORTB,Y         Write value to display
        JSR     DLY40           delay 40us
        XGDX                    place r back into D
        JSR     THOU            print 1000's digit
        JSR     HUND            print 100's digit
        JSR     TENZ            print 10's digit and 1's digit

        BCLR    PORTA,Y MUX3    reset MUX to read Ratio
        LDAB    PORTC,Y         read Ratio into B
        LDAA    #$00            zero out A
        XGDX                    move 16 bit version of Ratio to X

        LDAB    POSLSB          retrieve LSB part of A*ratio
        LDAA    #$19TIMES 25
        MUL                     25 * LSB part of A*ratio

        IDIV                    LSB / Ratio -> X; r -> D
        XGDX                    Place LSB / Ratio back into D
        JSR     TENZ            print 10's digit and 1's digit
        LDY     #UNIT1          pointer to units (Inches)
        JSR     PRINT2          print units
        JSR     DLY100          delay 100ms
        BRSET   PORTA,X $07 C2  if port A bit 0-2 still '111' re-display pos
        BCLR    PORTA,X MUX3    reset MUX
        BRA     C3              route back to main program
C2      JMP     C1              continue looping
C3      RTS                     end of subroutine
```

```
***
* MCUR:  Displays motor current in Amps with update every ? sec
***
MCUR    LDY    #TEXT3              pointer to motor current title
        JSR    PRINT               print title
C4      JSR    LINE2               setup 2nd line of display
        LDAA   ADR3,Y              Read motor current from A/D 2
        LDAB   #$05                Load current scaling factor
        MUL                        multiply A/D2 and scaling factor
        JSR    THOU                print 1000's digit
        JSR    HUND                print 100's digit
        JSR    TENZ                print 10's and 1's digits
        LDY    #UNIT2              pointer to units (Amps)
        JSR    PRINT2              print units
        JSR    DLY100              delay 100ms
        BRSET  PORTA,X $05 C5      if port A bit 0-2 still '101' re-display cur
        BRA    C6                  route back to main program
C5      JMP    C4                  continue looping
C6      RTS                        end of subroutine


***
* MCSET: Displays motor setpoint current in Amps, reads new current from dip switches, and
*        transmits the new current to the PWM 68HC11 at the push of the load button.
***
MCSET   LDY    #TEXT4              pointer to motor current title
        JSR    PRINT               print title
        BSET   PORTA,X MUX3        set MUX to read current dip
C10     JSR    LINE2               setup 2nd line of display
        LDAA   PORTC,Y             Read motor setpoint current from Port C
        LDAB   #$05                Load current scaling factor
        MUL                        multiply current and scaling factor
        JSR    THOU                print 1000's digit
        JSR    HUND                print 100's digit
        JSR    TENZ                print 10's and 1's digit
        LDY    #UNIT2              pointer to units (Amps)
        JSR    PRINT2              print units
        JSR    DLY100              delay 100ms
        BRSET  PORTA,X $07 C21
        BRA    C22
C21     JSR    SENDC
        BRA    C12
C22     BRSET  PORTA,X $83 C11     if port A bit 0-2 still '011' loop again
        BRA    C12                 route back to main program
C11     JMP    C10                 continue looping
C12     BCLR   PORTA,X MUX3        reset MUX
        RTS                        end of subroutine
```

```
***
* SENDC:  Sends setpoint current to PWM controller
***

SENDC   LDY    #TEXT5
        JSR    PRINT
        JSR    LINE2
        LDY    #TEXT6
        JSR    PRINT2
        LDAA   PORTC,X          Read motor setpoint current from Port C
SCI1    LDAB   SCSR,X           Read SCSR register
        BITB   #$80             Isolate the TDRE bit
        BEQ    SCI1             if Transmitter register not empty loop
        STAA   SCDR,X           store current to SCI data register - TxD
        JSR    DLY100
        JSR    DLY100
        JSR    DLY100
        JSR    LINE2
        LDY    #TEXT7
        JSR    PRINT2
        JSR    DLY100
        JSR    DLY100
        JSR    DLY100
        JSR    DLY100
        JSR    DLY100
        JSR    DLY100
        LDY    #TEXT8
        JSR    PRINT
        JSR    LINE2
        LDY    #TEXT9
        JSR    PRINT2
C20     BRCLR  PORTA,X $80 C20
        RTS                     end of subroutine


***
* DLY100 - Subroutine to delay 100mS (for E=2MHz)
***

DLY100  PSHX                    Save the value of X
        LDX    #$FFF0           20000 * 10 * 0.5uS = 100mS delay
D1LOOP  NOP                     [2] [# of clock cycles per instruction]
        NOP                     [2]
        DEX                     [3]
        BNE    D1LOOP           [3] continue for 4E20 times
        PULX                    Recover X value
        RTS                     end of subroutine
```

```
***
* DLY10 - Subroutine to delay 10mS (for E=2MHz)
***

DLY10   PSHX                    Save the value of X
        LDX   #$0D06            3334 * 6 * 0.5uS = 10mS delay
DLOOP   DEX                     [3] [# of clock cycles per instruction]
        BNE   DLOOP             [3] continue for 3334 times
        PULX                    Recover X value
        RTS                     end of subroutine


***
* DLY40 - Subroutine to delay 40uS (for E=2MHz)
***

DLY40   PSHX                    Save X value
        LDX   #$000D            13 * 6 * 0.5uS = 39uS delay
D4LOOP  DEX                     [3] decrement the X register
        BNE   D4LOOP            [3] continue for 13 times
        PULX                    Recover X value
        RTS                     end of subroutine
*
*
TEXT1   FCB   'B,'e,'l,'t,' ,'R,'a,'t,'i,'o,'/,'R,'e,'s,'.,$0
TEXT2   FCB   'A,'x,'i,'s,' ,'P,'o,'s,'i,'t,'i,'o,'n,$0
TEXT3   FCB   'M,'o,'t,'o,'r,' ,'C,'u,'r,'r,'e,'n,'t,$0
TEXT4   FCB   'S,'e,'t,'p,'o,'i,'n,'t,' ,'C,'u,'r,'r,'e,'n,'t,$0
TEXT5   FCB   'S,'e,'r,'i,'a,'l,' ,'D,'a,'t,'a,' ,'T,'r,'a,'n,'s,'f,'e,'r,$0
TEXT6   FCB   'I,'n,' ,'P,'r,'o,'g,'r,'e,'s,'s,$0
TEXT7   FCB   'C,'o,'m,'p,'l,'e,'t,'e,' ,' ,' ,$0
TEXT8   FCB   'P,'l,'e,'a,'s,'e,' ,'m,'o,'v,'e,' ,'s,'e,'l,'e,'c,'t,'o,'r,$0
TEXT9   FCB   't,'o,' ,'a,' ,'n,'e,'w,' ,'p,'o,'s,'i,'t,'i,'o,'n,'.,$0

UNIT1   FCB   ' ,'I,'n,'c,'h,'e,'s,$0
UNIT2   FCB   ' ,'A,'m,'p,'s,$0
RATIO0  FCB   '0,' ,'t,'o,' ,'0,' ,' ,'0,' ,'i,'n,'c,'h,'e,'s,$0
RATIO3  FCB   '3,' ,'t,'o,' ,'1,' ,' ,'0,'.,'0,'0,'2,'5,'i,'n,$0
RATIO2  FCB   '2,' ,'t,'o,' ,'1,' ,' ,'0,'.,'0,'0,'3,'7,'i,'n,$0
RATIO1  FCB   '1,' ,'t,'o,' ,'1,' ,' ,'0,'.,'0,'0,'7,'5,'i,'n,$0

        ORG   $C000  Sets the base address for the variables
RATX4   FDB   $0000
MATH1   FDB   $0000
POSLSB  FCB   $00
```

## Software listing for the 'PWM Controller'

```
*           File:               PWM.asm
*           Date:               8/10/94
*           Version:            2.04
*           Programmer:         Mark Holdaway
*           Contains:           Interface/controller from user interface to H-Drive.  Receives
*                               current setpoint from serial  port and generates two PWM signals
*                               and regulates current.
*
*           Hardware Connections from 68HC11 to DAC:
*                               Data out bits 0-7   PORTB   bits 0-7   (42-35)
*                               Enable-Data Latch   STRB               (6)
*
*           Hardware Connections from |A-B| to 68HC11:
*                               |A-B| bits 0 - 7    PORTC   bits 0-7   (9-16)
*                               |A-B| bits 8 - 11   PORTE   bits 4-7   (44, 46, 48, 50)
*                               A > B sign bit      PORTE   bit 3      (49)
*
*           Hardware Connections from 68HC11 to H-Drive:
*                               PWM1 output line    PORTA   bit 4      (30)
*                               PWM2 output line    PORTA   bit 5      (29)
*
*           Hardware Connections from User I/O 68HC11 to PWM 68HC11:
*                                RxD - SCI input    PORTD   bit 1      (21)
*
**Equates - Registers will be addressed with Ind,X mode
*
REGBAS  EQU     $1000       Starting address for register block
PORTA   EQU     $00         Data register Port A
PORTB   EQU     $04         Output Port B
PORTC   EQU     $03         Input/Output Port C
PORTE   EQU     $0A         Input Port E
PIOC    EQU     $02         Parallel I/O Control Register
BAUD    EQU     $2B         SCI BAUD rate control register
SCCR1   EQU     $2C         SCI control register 1
SCCR2   EQU     $2D         SCI control register 2
SCSR    EQU     $2E         SCI status register
SCDR    EQU     $2F         SCI data register
PWM     EQU     $30         Mask for PA4 and PA5
PWM1    EQU     $10         Mask for PA4
PWM2    EQU     $20         Mask for PA5
```

```
* Program begins here

        ORG     $C000           Prog starts in EVB RAM at $C000
START   LDS     #$0047          Load stack pointer for user stack on EVB
        LDX     #REGBAS         Pointer to register block
        BCLR    PORTA,X PWM     turn both PWM lines off
        LDAA    #$02            set strobe B to active low
        STAA    PIOC,X          store in parallel I/O control reg
        LDAA    #$00            initial current value on power-up
        STAA    PORTB,X         reset DAC current reference to 0
        LDAA    #$30            data to set highest baud to 9600
        STAA    BAUD,X          set baud rate to 9600 for E = 2.0 MHz
        LDAA    #$00            data to set mode to:
        STAA    SCCR1,X         1 start, 8 data, 1 stop bit
        LDAA    #$0C            data to enable the transmitter
        STAA    SCCR2,X         and receiver
        LDD     #$0000          data initialization
        STAA    FILL1           statements for FILL1,
        STAA    VREF            FILL2, and VREF with
        STAA    FILL2           the value of 0
        STD     MAGAB           Set MAGAB to 0
LOOP    LDAA    SCSR,X          (4) - Load SCI status register
        ANDA    #$20            (2) - isolate and test RDRF bit
        BEQ     SCI             (3) - if receiver's not full, don't read
        LDAA    SCDR,X          (4) - read SCI data register
        STAA    VREF            (4) - store the new current value in VREF

SCI     BRCLR PORTE,X $08 UP    (7) - if PE3=1 PWM1=1 else PWM2=1
        BSET    PORTA,X PWM1    (7) - set PWM1 line high
        BRA     UPDWN           (3) - jump past opposite command option
UP      BSET    PORTA,X PWM2    (7) - Set PWM2 line high
UPDWN   LDX     FILL1           (3) - Load Vref in as delay variable
        JSR     DLY             ( ) - jump to delay statement

        LDAB    PORTC,X         (4) - Load mag of |A-B| least sig 8 bits
        LDAA    PORTE,X         (4) - Load mag of |A-B| most sig 4 bits
        RORA                    (2) - Move value in A 1 bit lower
        RORA                    (2) - Move value in A 1 bit lower
        RORA                    (2) - Move value in A 1 bit lower
        RORA                    (2) - Move value in A 1 bit lower
        ANDA    #$0F            (2) - Mask off the lower 4 bits of Port E
        STD     MAGAB           ( ) - Save value of A-B into MAGAB
```

```
        LDAB   VREF          (4) - Load value of Vref into B reg
        RORB                 (2) - Move value in B 1 bit lower
        RORB                 (2) - Move value in B 1 bit lower
        ANDB   #$3F          (2) - Mast off the upper 2 bits of B reg
        LDAA   #$00          ( ) - Load 0 into reg A to form reg D
        CPD    MAGAB         ( ) - Compare Vref/4 and |A-B|
        BHS    LPOW          (3) - IF mag A-B/5 => Vref branch to FPOW
        LDAA   VREF          (4) - store full value of Vref to DAC
        STAA   PORTB,X       (4) - write Vref value to DAC
        BRA    END           (3) - re-run main program loop
LPOW    LDD    MAGAB         ( ) - Load value of MAGAB into port D
        STAB   PORTB,X       (4) - Store fractional value of Vref to DAC

END     BCLR   PORTA,X PWM   (7) - Set PWM line low (turn drive off)
        LDX    FILL1         (3) - Load Vref in as delay variable
        JSR    DLY           ( ) - jump to delay statement
        BRA    LOOP          (3) - re-run main program loop

***
* DLY - Subroutine to delay 1.0 mS (for E=2MHz)
***
DLY     INX                  [3] X * 10 * 0.5uS = X * 5.0 uS delay
DL      NOP                  [2]
        NOP                  [2]
        DEX                  [3] [# of clock cycles per instruction]
        BNE    DL            [3] continue for 7 times
        LDX    #REGBAS
        RTS                  end of subroutine


        ORG    $000A
FILL1   FCB    $0
VREF    FCB    $0
FILL2   FCB    $0
MAGAB   FDB    $0000
```

# ACSL simulation model

```
PROGRAM DC servo motor drive system
   INITIAL
      constant TSTART=0.0, TSTOP=7.0
      CINTERVAL CINT=0.007
      " 4A-1.569V, 5A-1.96V, 6A-2.35V, 7A-2.745V & 8A-3.14V "
      constant VLIM=2.745
      constant KB=0.027284, KT=3.8, K1=0.0784, K2=1.111, KO=0.00375
      constant TL=0.0, VPS=35.0, ON=0.0, DIR=0.0
      constant RC=0.008, Req=1.6, Bm=0.115
      constant freq=2400.0, dist=2400.0, load=5.0, hit=-10.0, hloc=3.5
      Js = 1.1066e-3 + (3.0 + load)/482.0
   END $ "of Initial"

   DERIVATIVE
      "inv(vlim/5 * 255 * 5.0uS * 2)--> 714 Hz Square Wave"
      PWM  = PULSE(0.0, 0.0014, 0.0007)
      TL   = (hit*16*0.2388)*(STEP(hloc)-STEP(hloc+(2*cint)))

      PCR  = freq*T
      PC   = BOUND(0.0, dist, PCR)
      PREF = KO*PC

      E    = PC - STP

      PROCEDURAL(ON=VIA, E)
         IF(E.LT.0) DIR=-1.0
         IF(E.GE.0) DIR=1.0
         VREF = BOUND(0.0, VLIM, ABS(K1*E))
         VRF2 = 0.635*VREF
         VXX  = ABS(VIA)
         IF(VXX.GE.VREF) ON=0.0
         IF(VXX.LT.VREF.AND.VIA.GT.VRF2) ON=ON
         IF(VXX.LE.VRF2) ON=1.0
      END $" OF PROCEDURAL "

      EA   = VPS*PWM*DIR*ON
      EV   = EA - KB*WM
      IA   = EV/REQ

      Vx   = 0.5*3.091*REALPL(RC, IA, 0.0)
      VIA  = BOUND(-12.0, 12.0, Vx)
      TM   = KT*IA
      TR   = TM - TL
      WM   = (REALPL(Js/Bm, TR, 0.0))/Bm
      THM  = 57.296*INTEG(WM, 0.0)
      STP  = K2*THM
      POS  = KO*STP
      termt(T.GE.TSTOP)
   END $ "of Derivative"
END $ "of Program"
```

# Vita

Mark Holdaway was born in Oak Ridge, TN in 1967. He received a BS in Electrical Engineering from Tennessee Technological University in 1990. In December of 1994 he received a Master of Science degree in Electrical Engineering from the University of Tennessee at Knoxville.